Information and Computation 209 (2011) 637–663

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

Verification of reactive systems via instantiation of Parameterised Boolean Equation Systems

B. Ploeger, J.W. Wesselink, T.A.C. Willemse *

Design and Analysis of Systems Group, Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Received 14 September 2009
Available online 9 December 2010

Keywords:
Model checking
Verification
Modal μ-calculus
Parameterised Boolean Equation Systems

ABSTRACT

Verification problems for finite- and infinite-state processes, like model checking and equivalence checking, can effectively be encoded in Parameterised Boolean Equation Systems (PBESs). Solving the PBES then solves the encoded problem. The decidability of solving a PBES depends on the data sorts that occur in the PBES. We describe a pragmatic methodology for solving PBESs, *viz.*, by attempting to instantiate them to the sub-fragment of Boolean Equation Systems (BESs). Unlike solving PBESs, solving BESs is a decidable problem. Based on instantiation, verification using PBESs can effectively be done fully automatically in most practical cases. We demonstrate this by solving several complex verification problems using a prototype implementation of our instantiation technique. In addition, practical issues concerning this implementation are addressed. Furthermore, we illustrate the effectiveness of instantiation as a transformation on PBESs when solving verification problems involving systems of infinite size.

1. Introduction

In the field of verification, the aim is to prove the correctness of sequential and concurrent computer programs, in particular reactive systems. Formal methods for system verification include model checking and equivalence checking. Given a model of a system and a desired property expressed as a formula in some temporal logic, a model checker decides whether the model satisfies the formula and, hence, whether the system has the property. In equivalence checking, the correctness of a model is established by proving that the model is behaviourally equivalent to another, trusted model (*e.g.* a specification) using an appropriate notion of equivalence.

For finite models, model checking and equivalence checking are decidable for a wide variety of temporal logics and equivalences. However, explicitly represented models tend to become extremely large in practice, by which automated verification quickly breaks down – a problem known as the *state space explosion problem*. Such models are often generated from concise, implicit specifications that symbolically represent the state space. These specifications are written in higher-level languages that even allow for infinite models to be represented in a finite, concise manner. Because of this property, automated verification is ideally done on this symbolic level to circumvent the state space explosion problem.

In this paper, we advocate a methodology for the verification of finite- and infinite-state systems on that symbolic level using *Parameterised Boolean Equation Systems* (PBESs) [1,2]. These sequences of fixpoint equations have emerged as a versatile vehicle for studying and solving verification problems like model checking [3,4], equivalence checking [5] and static analysis of code [6]. We describe PBESs in more detail in Section 3. An overview of our verification approach is shown

* Corresponding author. Fax: +31 40 2468508.
E-mail addresses: S.C.W.Ploeger@tue.nl (B. Ploeger), J.W.Wesselink@tue.nl (J.W. Wesselink), timw@win.tue.nl (T.A.C. Willemse).

0890-5401/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2010.11.025

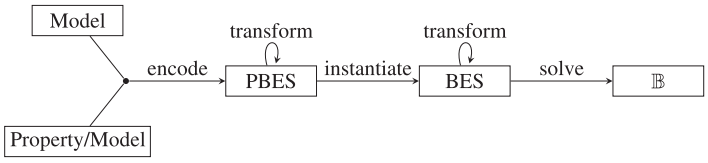


Fig. 1. In our methodology, a verification problem is represented as a PBES which is instantiated to a BES and subsequently solved.

in Fig. 1. For model checking, a model and a property can be encoded in a PBES such that the solution to the model-checking problem corresponds with the solution to the equation system. This encoding can be done fully automatically for the first-order modal μ-calculus and infinite-state models [3,4]. For various bisimulations, the equivalence-checking problem on two possibly infinite models can be encoded in a PBES automatically, such that the solution to the problem corresponds with that of the PBES [5].

Solving PBESs is generally undecidable, much like the problems that can be encoded in them. The outlook, however, is not that bleak: practical applications have illustrated that a pragmatic approach can lead to promising results [4]. A number of techniques have been developed for transforming PBESs into simpler ones for which a solution may be obtained more easily, like *symbolic approximation* [4] and *pattern matching* [1]. In our envisaged verification process, *instantiation* [7] plays an important role. This technique aims to eliminate data from a PBES and takes inspiration from the algorithm that is proposed in [2] to construct and solve alternation-free PBESs. Our transformation can be applied *partially*, resulting in a simpler PBES, or *fully*, resulting in a *Boolean Equation System* (BES) [8] if all data sorts are finite, or an *Infinite BES* (IBES) [9] if all data sorts are countable. The instantiation technique is described and proven correct in Sections 4 and 5.

Full instantiation to a BES is particularly useful for automated verification: solving BESs is decidable (see *e.g.* [8]) and can be done efficiently in many practical applications, for instance by translating them to parity games and using state-of-the-art algorithms for solving these [10–12]. Also, recently developed transformations can reduce the size of a BES considerably, which can speed up the process of finding the solution [13,14]. When the PBES contains infinite, yet countable sorts, a finite BES may still be obtained by adopting an on-the-fly strategy in which only those parts of the IBES are generated that are relevant for the solution of a particular variable. We have implemented an on-the-fly version of our instantiation algorithm to allow for automated verification. In order to obtain a proper BES, any first-order predicates have to be removed from the PBES. Therefore, we have also implemented a procedure for the elimination of quantifiers, which is used by our instantiation tool. These procedures are described and proven correct in Section 6.

We illustrate the efficacy of our verification approach by several examples in Section 7. We encode two model checking problems on infinite-state systems from the literature in PBESs, and use partial instantiation to solve these PBESs, along with other solution techniques. To demonstrate the practical feasibility of verification using PBESs, we encode a variety of model checking and equivalence checking problems in PBESs, which are then solved using our instantiation tool and BES solution tools. All of this, from encoding to solving, is done fully automatically.

1.1. Related work

The current paper is an extended version of [7], with detailed proofs, additional examples and applications, and a discussion on implementation aspects.

Independently of the current paper and *ibid*, the approach taken in [2] has recently been implemented in a tool called Evaluator 4.0, see [15]. In essence, this tool employs alternation-free PBESs to solve the on-the-fly model-checking problem of Mcl formulas on finite systems. While the restriction to alternation-free PBESs has its consequences for the types of problems that can be verified, the underlying process of full instantiation, as implemented in [15], and also explained in [7] and the current paper, is similar in spirit and rooted in the same theory. In addition, our partial instantiation technique also allows one to manipulate PBESs encoding model-checking problems for infinite-state systems.

BESs, obtained using a full (on-the-fly) instantiation of a PBES, can be solved in different ways. Alternation-free BESs can be solved efficiently using, *e.g.* the Caesar_Solve library of CADP [16]. For alternating BESs, efficient algorithms based on Parity Games have been implemented, see *e.g.* [17] for a multi-core implementation of the *Small Progress Measures* algorithm due to Jurdziński [18], or the *bigstep* algorithm, due to Schewe [10], implemented in *e.g.* the PGSolver tool [19].

2. Preliminaries

Our main focus in this paper is on the automated verification of properties over processes with data. As a formal framework for specifying such processes, we use the process algebra mCRL2 [20]. Its basic process constructs are along the lines of ACP [21] and CCS [22], though its syntax is influenced mainly by ACP and μCRL [23]. The properties we consider are specified in a first-order extension of the modal μ-calculus (henceforth simply referred to as the *μ-calculus*), see *e.g.* [1,2]. Data are an integral part of both the μ-calculus and mCRL2.

2.1. Data

Throughout this article, we assume that data sorts represent non-empty data types (possibly containing uncountably many elements). As a convention, we write data sorts using letters D, E and F. In line with standard abstract data type theoretical approaches, we furthermore assume that a data sort specification consists of a sort declaration, constructor elements, operations and equations that state how the operations and constructors (and possibly other data sorts) are related.

For each countable data sort, we assume that each term can be written using unique *basic elements*; collectively, these basic elements make up the data sort. For a sort D, we write $v \in D$ to denote that v is a basic element of D and we use set notation to list the basic elements of D, *e.g.* $D = \{v_1, \ldots, v_n\}$. With every sort D we associate a semantic set $\mathbb{D}$ such that every syntactic term of sort D can be mapped to the element of $\mathbb{D}$ it represents. The set of basic elements of a countable sort D is isomorphic to the semantic set $\mathbb{D}$.

We have a set $\mathcal{D}$ of *data variables*, with typical elements d, d_1, . . ., and we assume that there is some data language that is sufficiently rich to denote all relevant *data terms*, such as for instance $3 + d_1 \leq d_2$. Syntactic substitution of a term e for every free occurrence of a variable d in a term t is denoted by $t[d := e]$. For a closed term t of sort D (denoted $t{:}D$), we assume an interpretation function $[\![t]\!]$ that maps t to the data element of $\mathbb{D}$ it represents. For open terms we use a *data environment* ε that maps each variable from $\mathcal{D}$ to a data element of the right sort. The interpretation of an open term t, denoted as $[\![t]\!]\varepsilon$ is given by $\varepsilon(t)$ where ε is extended to terms in the standard way. Throughout this paper, we use the following convention: for a (countable) set $\mathcal{V}$ of variables, a (possibly infinite) domain of values $\mathbb{V}$ and an environment $\theta{:}\mathcal{V} \to \mathbb{V}$, a variable $v \in \mathcal{V}$ and a value $w \in \mathbb{V}$, we write $\theta[v \mapsto w]$ for the environment θ', defined as $\theta'(v') = \theta(v')$ for all variables v' different from v and $\theta'(v) = w$. In effect, $\theta[v \mapsto w]$ stands for the environment θ where the variable v has been assigned the value w. For substitution on tuples we define $\theta[(d_1, \ldots, d_n) \mapsto (v_1, \ldots, v_n)]$ to be equivalent to the simultaneous substitution $\theta[d_1 \mapsto v_1, \ldots, d_n \mapsto v_n]$.

We furthermore assume the existence of a sort $B = \{\top, \bot\}$ representing the Booleans $\mathbb{B}$, and a sort $N = \{0, 1, \ldots\}$ representing the natural numbers $\mathbb{N}$. For these sorts, we assume the usual operators are available and we do not write constants or operators in the syntactic domain any different from their semantic counterparts. For example, we have $\mathbb{B} = \{\top, \bot\}$ and the syntactic operator $\_ \wedge \_{:}B \times B \to B$ corresponds to the usual, semantic conjunction $\_ \wedge \_{:}\mathbb{B} \times \mathbb{B} \to \mathbb{B}$.

2.2. Process descriptions

The language mCRL2 has a small number of basic operators and primitives. Processes are the main objects in the language. A set of parameterised actions *Act* is assumed; actions can be considered as functions from a data domain to a process. An action $a \in Act$ represents an atomic event, taking a number of data arguments. The process representing no behaviour, *i.e.* the process that cannot perform any actions is denoted δ. This constant is often referred to as *deadlock* or *inaction*. Note that the process a terminates successfully immediately after executing the action a, whereas the process $a \cdot \delta$ does not terminate successfully.

Processes are constructed using several operators. The main operators are alternative composition ($p + q$ for some processes p and q) and sequential composition ($p \cdot q$ for some processes p and q). Conditional behaviour is denoted using a ternary operator (we write $b \to p \diamond q$ when we mean process p if b holds and else process q). The process $b \to p$ serves as a shorthand for $b \to p \diamond \delta$, which represents the process p under the condition that b holds. Recursive behaviour is specified using equations. Data is intertwined with processes such that process variables can be considered as functions from a data domain to processes. Consider the following process.

$$X(n{:}N) = up \cdot X(n+1) + show(n) \cdot X(n) + (n > 0) \to down \cdot X(n-1)$$

The behaviour denoted by process $X(n)$ is the increasing and the decreasing of an internal counter n or showing its current value. Note that the *up* and *down* actions do not have parameters. For the formal exposition, however, it can be more convenient to assume that actions and processes have a single parameter. This assumption is easily justified, as we can assume the existence of a singleton data domain, together with adequate pairing and projection functions.

The last operator considered here is data-dependent alternative quantification (we write $\sum_{d:D} p$ to denote the alternatives of process p, dependent on some arbitrary datum d selected from the (possibly infinite) data domain D). The $\sum$-operator is best compared to *e.g.* input prefixing, but is more expressive (see *e.g.* [24]). As an example of the $\sum$-operator we consider a process that can set an internal counter to an arbitrary value, which can be read at will:

$$V(n{:}N) = read(n) \cdot V(n) + \sum_{n'{:}N} set(n') \cdot V(n')$$

A more complex notion of process composition is the parallel composition of processes. It consists of all possible interleavings of the action sequences of the involved processes, along with the synchronisation of some of their actions. For verification or analysis purposes, it is often convenient to eliminate parallelism in favour of sequential composition and (quantified) alternative composition. This can be done fully automatically for a practically sufficiently large set of process expressions, see [25]. A behaviour of a process can then be denoted as a state-vector of typed variables, accompanied by a set of condition–action–effect rules. Processes denoted in this fashion are called *Linear Process Equations*.

Definition 1 (*Linear Process Equations*)**.** A *Linear Process Equation* (LPE) is a parameterised equation taking the form

$$X(d{:}D) = \sum_{i:I} \sum_{e_i:D_i} c_i(d, e_i) \to a_i(f_i(d, e_i)) \cdot X(g_i(d, e_i))$$

where I is a finite index set; D and D_i are data domains; d and e_i are data variables; $a_i \in Act$ are actions with parameters of sort D_{a_i}; $f_i{:}D \times D_i \to D_{a_i}$, $g_i{:}D \times D_i \to D$ and $c_i{:}D \times D_i \to B$ are functions. The function f_i yields, on the basis of the current state d and the non-deterministically chosen value for the bound variable e_i, the parameter for an action a_i; the "next state" is encoded in the function g_i, and is determined on the basis of the current state and the bound variable e_i. The function c_i describes when action a_i can be executed. The data domain D is referred to as the *parameter set* of X.

In this paper, we restrict ourselves to the use of processes that do not terminate successfully; note that this still allows for processes that terminate *unsuccessfully* (represented by δ). Including termination into our theory does not pose any theoretical challenges, but is omitted in our exposition for brevity. In the remainder of this paper, we use the LPE-notation as a vehicle for our exposition of the theory and practice. The (operational) semantics of an LPE can be defined in terms of the labelled transition system that it induces.

Definition 2 (*Transition system of an LPE*)**.** The *labelled transition system* of a Linear Process Equation as defined in Definition 1 is a quadruple $M = \langle S, \Sigma, \to, s_0 \rangle$, where

- $S = \mathbb{D}$ is the (possibly infinite) set of *states*;
- $\Sigma = \{a_i(\llbracket d_{a_i} \rrbracket) \mid i \in I \wedge a_i \in Act \wedge d_{a_i} \in D_{a_i}\}$ is the (possibly infinite) set of labels;
- $\to = \{(\llbracket d \rrbracket, a_i(\llbracket f_i(d, e_i) \rrbracket), \llbracket g_i(d, e_i) \rrbracket) \mid d \in D \wedge i \in I \wedge a_i \in Act \wedge e_i \in D_i \wedge \llbracket c_i(d, e_i) \rrbracket\}$ is the *transition relation*;
- $s_0 = \llbracket d_0 \rrbracket$, for a given $d_0 \in D$, is the *initial state*.

2.3. First-order modal μ-calculus

The logic used for specifying properties is based on the standard modal μ-calculus [26], extended with data variables, quantifiers and parameterised fixpoints, see, *e.g.* [1–3]. In this section, we review its syntax and semantics, and we demonstrate its use by means of several small examples.

Definition 3. The μ-calculus is given by the following BNF grammar, where φ represents a *state formula* and α represents an *action formula*:

$$\begin{aligned}
\varphi \;&::=\; b \mid Z(e_1, \ldots, e_n) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid [\alpha]\varphi_1 \mid \\
&\qquad \forall d{:}D.\, \varphi \mid \nu Z(d_1{:}D_1 = e_1, \ldots, d_n{:}D_n = e_n).\, \varphi \\
\alpha \;&::=\; a(e) \mid b \mid \neg\alpha_1 \mid \alpha_1 \wedge \alpha_2 \mid \forall d{:}D.\, \alpha_1
\end{aligned}$$

b is an expression of sort B, possibly containing data variables d of a set of variables $\mathcal{D}$; e is a data expression (possibly containing data variables d of the set $\mathcal{D}$) of type D; $Z{:}D_1 \times \cdots \times D_n \to B$ is a sorted recursion variable from a set of sorted predicate variables $\mathcal{P}$. Expressions of the form $(\nu Z(d_1{:}D_1 = e_1, \ldots, d_n{:}D_n = e_n).\, \varphi)$ are subject to the restriction that any free occurrence of Z in φ must be within the scope of an even number of negation symbols. Finally, $a(e)$ is an arbitrary action from the set $\mathcal{A}ct$, parameterised with expression e.

For simplicity, and without loss of generality, we restrict to predicate variables of arity ≤ 1 in the theoretical considerations that follow, writing $Z(e)$ instead of $Z(e_1, \ldots, e_m)$ and $\nu Z(d{:}D{=}e).\varphi$ instead of $\nu Z(d_1{:}D_1{=}e_1, \ldots, d_n{:}D_n{=}e_n).\varphi$. We restrict ourselves to μ-calculus formulae given in *Positive Normal Form* (PNF). This means that negation only occurs at the lowest level, and, in addition, all bound variables are distinct. Note that every μ-calculus formula can be converted into PNF by suitable α-renaming and relying on logical rules such as *De Morgan*.

The semantics of μ-calculus formulae is defined by means of an interpretation over a labelled transition system M that is induced by an LPE (recall Definition 2). Since μ-calculus expressions can be open terms, the semantics is defined in the context of a given environment.

Definition 4. Let $M = \langle S, \Sigma, \to, s_0 \rangle$ be an LTS, induced by an LPE. Let $\varepsilon{:}\mathcal{D} \to \mathbb{D}$ be a data environment, and $\rho{:}\mathcal{P} \to (\mathbb{D} \to 2^S)$ be a sorted predicate environment. The interpretation of a μ-calculus formula φ, denoted by $\llbracket\varphi\rrbracket\rho\varepsilon$, is given in the context of ε, ρ and the LTS M:

$$\begin{aligned}
\llbracket b \rrbracket \rho\varepsilon \quad &\triangleq \begin{cases} S \text{ if } \varepsilon(b) \\ \emptyset \text{ otherwise} \end{cases} \\
\llbracket Z(e) \rrbracket \rho\varepsilon \;&\triangleq \rho(Z)(\epsilon(e))
\end{aligned}$$

$$
\begin{aligned}
&[\![\neg\varphi]\!]\rho\varepsilon && \triangleq S \setminus [\![\varphi]\!]\rho\varepsilon \\
&[\![\varphi_1 \wedge \varphi_2]\!]\rho\varepsilon && \triangleq [\![\varphi_1]\!]\rho\varepsilon \cap [\![\varphi_2]\!]\rho\varepsilon \\
&[\![[\alpha]\varphi]\!]\rho\varepsilon && \triangleq \{w \in S \mid \forall w' \in S\, \forall a \in \Sigma\, (w \xrightarrow{a} w' \wedge a \in [\![\alpha]\!]\varepsilon) \Rightarrow w' \in [\![\varphi]\!]\rho\varepsilon\} \\
&[\![\forall d{:}D.\varphi]\!]\rho\varepsilon && \triangleq \textstyle\bigcap_{v \in \mathbb{D}} [\![\varphi]\!]\rho(\varepsilon[d \mapsto v]) \\
&[\![\nu Z(d{:}D{=}e).\ \varphi]\!]\rho\varepsilon && \triangleq (\nu\Phi_{\rho\varepsilon})(\varepsilon(e))
\end{aligned}
$$

where we define

$$\Phi_{\rho\varepsilon} \triangleq \lambda F{:}\mathbb{D} \to 2^S.\lambda v{:}\mathbb{D}.[\![\varphi]\!](\rho[Z \mapsto F])(\varepsilon[d \mapsto v])$$

Note that the ordered set $\langle [\mathbb{D} \to 2^S], \sqsubseteq \rangle$ is a complete lattice, where $[\mathbb{D} \to 2^S]$ is the set of functions from $\mathbb{D}$ to subsets of S and $\sqsubseteq$ is defined as $f \sqsubseteq g$ iff for all $v \in \mathbb{D}$, we have $f(v) \subseteq g(v)$. The interpretation of fixpoint expressions is then justified since the functionals $\Phi_{\rho\varepsilon}$ are monotonic over this lattice, see [3]. From Tarski's Theorem [27], the existence and uniqueness of fixpoints readily follows.

The action formulae that are present in the modalities in the state formulae are interpreted as follows. Let α be an action formula; we denote its interpretation by $[\![\alpha]\!]\varepsilon$, which we define inductively as:

$$
\begin{aligned}
&[\![b]\!]\varepsilon \triangleq \begin{cases} \Sigma \text{ if } \varepsilon(b) \text{ holds} \\ \emptyset \text{ otherwise} \end{cases} && [\![\neg\alpha]\!]\varepsilon && \triangleq \Sigma \setminus [\![\alpha]\!]\varepsilon \\
&[\![a(e)]\!]\varepsilon \triangleq \{a(\varepsilon(e))\} && [\![\alpha_1 \wedge \alpha_2]\!]\varepsilon && \triangleq [\![\alpha_1]\!]\varepsilon \cap [\![\alpha_2]\!]\varepsilon \\
& && [\![\forall d{:}D.\alpha]\!]\varepsilon && \triangleq \textstyle\bigcap v \in D\, [\![\alpha]\!]\varepsilon[d \mapsto v]
\end{aligned}
$$

We introduce the following standard abbreviations for μ-calculus formulae φ, action formulae α and (both μ-calculus formulae and action formulae) ψ.

$$
\begin{aligned}
&\bot && \triangleq \neg\top \\
&(\psi_1 \vee \psi_2) && \triangleq \neg(\neg\psi_1 \wedge \neg\psi_2) \\
&\langle\alpha\rangle\varphi && \triangleq \neg[\alpha]\neg\varphi \\
&\exists d{:}D.\psi && \triangleq \neg\forall d{:}D.\neg\psi \\
&\mu Z(d{:}D = e).\varphi && \triangleq \neg\nu Z(d{:}D{=}e).\ \neg\varphi[Z := \neg Z]
\end{aligned}
$$

Most of the constructs in the μ-calculus presented here are fairly straightforward. Using the action formulae, one can conveniently specify properties for systems with infinite sets of actions, as demonstrated by the following example.

Example 1. Suppose a system has a parameterised action a of sort naturals. Typically, in that case, the underlying transition system M has a set of labels Σ of infinite size. In order to specify the process does not exhibit any deadlocks, it suffices to specify the formula $\nu X.[\top]X \wedge \langle\top\rangle\top$.

The use of parameterised fixed points may appear awkward at first, but their presence allows for conveniently and compactly specifying extremely complex problems. Below, two examples of their uses are provided.

Example 2. A parameterised fixed point can be used to "memorise" vital process information. Consider the below formula:

$$\exists n{:}N.\ (\nu Z(i{:}N = n).\ \langle a(i)\rangle Z(i+1))$$

The counter i is used to remember the last value that is communicated via action a. In essence, the above formula holds in any process that can perform an infinite sequence $a(n)\ a(n+1)\ldots$, for some value n.

Example 3. A slightly more elaborate formula is given below. Let $\mathcal{L}(N)$ denote a list of natural numbers. Assume that we have the following operations on $\mathcal{L}(N)$: $m \vdash l$ denotes prefixing of list l by element m, and, for a non-empty list l, $\mathsf{head}(l)$ yields the head of l and $\mathsf{tail}(l)$ yields the tail of l.

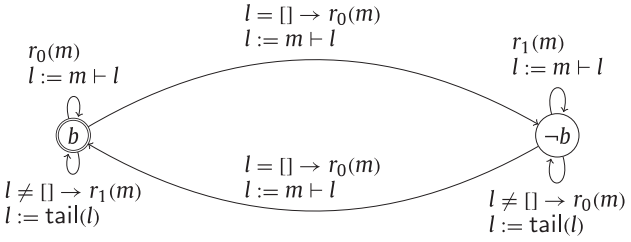
$$
\begin{aligned}
&(\nu Z(l{:}\mathcal{L}(N) = [], b{:}B = \top). \\
&\quad \forall m{:}N.\ \ [r_0(m)]((b \implies Z(m \vdash l, b)) \wedge \\
&\qquad\qquad ((\neg b \wedge l \neq []) \implies (Z(\mathsf{tail}(l), b) \wedge m = \mathsf{head}(l))) \wedge \\
&\qquad\qquad ((\neg b \wedge l = []) \implies Z(m \vdash l, \neg b)))
\end{aligned}
$$

$$\wedge\ [r_1(m)](\ (\neg b \implies Z(m \vdash l, b)) \wedge$$
$$((b \wedge l \neq []) \implies (Z(\mathsf{tail}(l), b) \wedge m = \mathsf{head}(l))) \wedge$$
$$((b \wedge l = []) \implies Z(m \vdash l, \neg b)))$$
$$)$$

The above formula expressed that the sequence of values that is read via r_0 and r_1 has to correspond as long as values can be read via these actions. The Boolean variable b indicates whether the stream of values read via r_0 is ahead of r_1 (when b holds), or is lagging behind (when b does not hold). List l remembers the differences in the streams. Whenever b holds and a value m is read via r_0, it is added to the list l, indicating that m should still be read via r_1. If, on the other hand, a value m is read via r_1, b holds and m is at the top of list l, we are confronted with two situations:

1. $\mathsf{tail}(l)$ is not empty, in which case the stream r_1 is still all values in $\mathsf{tail}(l)$ behind,
2. $\mathsf{tail}(l)$ is empty, in which case the stream of values read via r_0 and r_1 are in full agreement.

Intuitively, the following graph visualises the control aspects of the property:



We finish this section with two typical verification problems encountered in the literature; both problems can be solved within the framework of *Parameterised Boolean Equation Systems*, the framework introduced in Section 3.

1. The *equivalence checking problem* is the problem of deciding whether the underlying transition systems of two processes X and Y are related via an equivalence relation such as *strong bisimilarity* [28,29] or *branching bisimilarity* [30].
2. The *local μ-calculus model checking problem* is the problem of deciding whether a given μ-calculus formula φ holds in the initial state of the underlying transition system M of a given process X, *i.e.* whether $s_0 \in [\![\varphi]\!]\rho\varepsilon$ for given environments ρ and ε.

Note that in contrast to the *local* model checking problem, the *global* model checking problem checks for *all* states of a transition system whether they satisfy a given formula.

3. Equation systems

3.1. Parameterised Boolean Equation Systems

Parameterised Boolean Equation Systems are sequences of fixpoint equations where each equation is of the following form:

$$\sigma X(d_1{:}D_1, \ldots, d_n{:}D_n) = \varphi.$$

The left-hand side of each equation consists of a *fixpoint symbol* $\sigma \in \{\mu, \nu\}$, where μ indicates a least and ν a greatest fixpoint, and a *predicate variable* $X{:}D_1 \times \cdots \times D_n \to B$ (from a set of variables $\mathcal{X}$) that depends on n data variables $d_1, \ldots, d_n$ of possibly infinite sorts $D_1, \ldots, D_n$. If $n = 0$ we have $X{:}B$ and we call X a *proposition variable*. The right-hand side of each equation is a *predicate formula* containing data terms, Boolean connectives, quantifiers over (possibly infinite) data domains and data and predicate variables. For simplicity and without loss of generality, in this section we restrict to predicate variables of arity ≤ 1 in the theoretical considerations that follow.

Definition 5 (*Predicate formulae*)**.** *Predicate formulae* φ are defined by the following grammar:

$$\varphi ::= b \mid X(e) \mid \varphi \oplus \varphi \mid \mathsf{Q}d{:}D.\varphi$$

where $\oplus \in \{\wedge, \vee\}$, $\mathsf{Q} \in \{\forall, \exists\}$, b is a data term of sort B, X is a predicate variable, d is a data variable of sort D and e is a data term.

Note that negation does not occur in predicate formulae, except as an operator in data terms. As a notational convenience, we use the operators $\oplus$ and Q throughout this paper when the exact operator is of lesser importance. Also, we call a predicate

formula φ *closed* if no data variable in φ occurs freely. We now formalise the notion of a *Parameterised Boolean Equation System*.

Definition 6 (*Parameterised Boolean Equation System*)**.** A *parameterised Boolean Equation System (PBES)* is inductively defined as follows:

- ϵ is the empty PBES;
- for every PBES $\mathcal{E}$, $(\sigma X(d{:}D) = \varphi)\ \mathcal{E}$ is also a PBES, where $\sigma \in \{\mu, \nu\}$ is a fixpoint symbol, $X{:}D \to B$ is a predicate variable and φ is a predicate formula.

The set of predicate variables that occur in a predicate formula φ, denoted by $\mathsf{occ}(\varphi)$, is defined recursively as follows, for any formulae φ_1, φ_2:

$$\mathsf{occ}(b) \triangleq \emptyset \qquad \mathsf{occ}(X(e)) \triangleq \{X\}$$
$$\mathsf{occ}(\varphi_1 \oplus \varphi_2) \triangleq \mathsf{occ}(\varphi_1) \cup \mathsf{occ}(\varphi_2) \qquad \mathsf{occ}(\mathsf{Q}d{:}D.\varphi_1) \triangleq \mathsf{occ}(\varphi_1).$$

For any PBES $\mathcal{E}$, the set of *binding predicate variables*, $\mathsf{bnd}(\mathcal{E})$, is the set of variables occurring at the left-hand side of some equation in $\mathcal{E}$. The set of *occurring predicate variables*, $\mathsf{occ}(\mathcal{E})$, is the set of variables occurring at the right-hand side of some equation in $\mathcal{E}$. The set of predicate variables occurring anywhere in $\mathcal{E}$ is denoted by $\mathsf{var}(\mathcal{E})$. Formally, we define:

$$\mathsf{bnd}(\epsilon) \triangleq \emptyset \qquad \mathsf{bnd}((\sigma X(d{:}D) = \varphi)\ \mathcal{E}) \triangleq \mathsf{bnd}(\mathcal{E}) \cup \{X\}$$
$$\mathsf{occ}(\epsilon) \triangleq \emptyset \qquad \mathsf{occ}((\sigma X(d{:}D) = \varphi)\ \mathcal{E}) \triangleq \mathsf{occ}(\mathcal{E}) \cup \mathsf{occ}(\varphi)$$
$$\mathsf{var}(\mathcal{E}) \triangleq \mathsf{bnd}(\mathcal{E}) \cup \mathsf{occ}(\mathcal{E}).$$

A PBES $\mathcal{E}$ is said to be *well-formed* iff every binding predicate variable occurs at the left-hand side of precisely one equation of $\mathcal{E}$. Thus, $(\nu X = \top)(\mu X = \bot)$ is not a well-formed PBES. We only consider well-formed PBESs in this paper.

We say a PBES $\mathcal{E}$ is *closed* whenever $\mathsf{occ}(\mathcal{E}) \subseteq \mathsf{bnd}(\mathcal{E})$ and if $\mathcal{E}$ is not closed, we say $\mathcal{E}$ is *open*. An equation $\sigma X(d{:}D) = \varphi$ is called *data-closed* if the set of data variables that occur freely in φ is either empty or $\{d\}$. A PBES is called *data-closed* iff each of its equations is data-closed. We say an equation $\sigma X(d{:}D) = \varphi$ is *solved* if $\mathsf{occ}(\varphi) = \emptyset$, and a PBES $\mathcal{E}$ is *solved* iff each of its equations is solved.

Finally, we give the denotational semantics of predicate formulae and PBESs. Predicate formulae are interpreted in a context of a data environment ε and a *predicate environment* $\eta{:}\mathcal{X} \to (\mathbb{D} \to \mathbb{B})$.

Definition 7 (*Semantics of predicate formulae*)**.** Let ε be a data environment and $\eta{:}\mathcal{X} \to (\mathbb{D} \to \mathbb{B})$ be a predicate environment. The *interpretation* $[\![\varphi]\!]\eta\varepsilon$ that maps a predicate formula φ to $\top$ or $\bot$, is inductively defined as follows:

$$\begin{aligned}
[\![b]\!]\eta\varepsilon &\triangleq [\![b]\!]\varepsilon \\
[\![X(e)]\!]\eta\varepsilon &\triangleq \eta(X)([\![e]\!]\varepsilon) \\
[\![\varphi_1 \oplus \varphi_2]\!]\eta\varepsilon &\triangleq [\![\varphi_1]\!]\eta\varepsilon \oplus [\![\varphi_2]\!]\eta\varepsilon \\
[\![\mathsf{Q}d{:}D.\varphi]\!]\eta\varepsilon &\triangleq \mathsf{Q}v \in \mathbb{D}\,.\,[\![\varphi]\!]\eta(\varepsilon[d \mapsto v]).
\end{aligned}$$

The predicate formula φ in an equation $\sigma X(d{:}D) = \varphi$ must be interpreted as a fixpoint over the set of functions with domain $\mathbb{D}$ and co-domain $\mathbb{B}$. Note that the existence of such fixpoints follows from the following observations. The variable d, which may occur free in φ, is effectively used as a formal, syntactic function parameter. Semantically, this is achieved by associating the interpretation of the predicate formula φ to the functional $(\lambda v \in \mathbb{D}.\ [\![\varphi]\!]\eta\varepsilon[d \mapsto v])$, which relies on the data environment to assign specific values to variable d. The set of (total) functions $f{:}\mathbb{D} \to \mathbb{B}$, denoted by $\mathbb{B}^{\mathbb{D}}$ can be equipped with an ordering $\sqsubseteq$, defined as follows:

$$f \sqsubseteq g \triangleq \forall d \in \mathbb{D}\,.\,f(d) \Rightarrow g(d).$$

The set $(\mathbb{B}^{\mathbb{D}}, \sqsubseteq)$ is a complete lattice. The functional $(\lambda v \in \mathbb{D}.\ [\![\varphi]\!]\eta\varepsilon[d \mapsto v])$ can be turned into a predicate formula transformer by employing the predicate environment η in a similar manner as the data environment is used to turn a predicate formula into a functional. Assuming that the domain of the predicate variable X is of sort D, the functional $(\lambda v \in \mathbb{D}.\ [\![\varphi]\!]\eta\varepsilon[d \mapsto v])$ yields the following predicate formula transformer:

$$\lambda g \in \mathbb{B}^{\mathbb{D}}.\ (\lambda v \in \mathbb{D}.\ [\![\varphi]\!]\eta[X \mapsto g]\varepsilon[d \mapsto v]).$$

The resulting predicate formula transformer is monotone over the complete lattice $(\mathbb{B}^{\mathbb{D}}, \sqsubseteq)$. As a corollary of Tarski's fixpoint Theorem [27], the existence of least and greatest fixpoints of the predicate formula transformers is guaranteed. We denote the extremal fixpoints of the above predicate formula transformers as follows:

$$\sigma g \in \mathbb{B}^{\mathbb{D}}.\ (\lambda v \in \mathbb{D}.\ [\![\varphi]\!]\eta[X \mapsto g]\varepsilon[d \mapsto v]).$$

Definition 8 (*Solution of a PBES*)**.** The *solution of a PBES* in the context of a predicate environment η and a data environment ε is inductively defined as follows, for any PBES $\mathcal{E}$:

$$\begin{array}{lcl} [\![\epsilon]\!]\eta\varepsilon & \stackrel{\Delta}{=} & \eta \\ [\![(\sigma X(d{:}D) = \varphi)\mathcal{E}]\!]\eta\varepsilon & \stackrel{\Delta}{=} & [\![\mathcal{E}]\!](\eta[X \mapsto \sigma f \in \mathbb{B}^{\mathbb{D}}.\ \lambda v{\in}\mathbb{D}.[\![\varphi]\!]([\![\mathcal{E}]\!]\eta[X \mapsto f]\varepsilon)\varepsilon[d \mapsto v]])\varepsilon. \end{array}$$

The solution of a PBES prioritises the fixpoint signs of equations that come first over the signs of equations that follow. In that sense, the solution is sensitive to the order of equations in a PBES. Moreover, the solution of a PBES only assigns functions to the *binding* variables of that PBES; other predicate variables are left unmodified. This follows from the following lemma:

Lemma 1. *Let $\mathcal{E}$ be an arbitrary PBES. Then:*

$$\forall X \notin \mathsf{bnd}(\mathcal{E})\ .\ \forall \eta, \varepsilon\ .\ [\![\mathcal{E}]\!]\eta\varepsilon(X) = \eta(X).$$

Proof. The proof is by induction on the length of equation system $\mathcal{E}$. If $\mathcal{E}$ is of length 0 then the equivalence holds vacuously. Suppose $\mathcal{E}$ is of length $m+1$, and for all $\mathcal{E}'$ of length m, we have:

$$\forall X \notin \mathsf{bnd}(\mathcal{E}')\ .\ \forall \eta, \varepsilon\ .\ [\![\mathcal{E}']\!]\eta\varepsilon(X) = \eta(X). \tag{IH}$$

Necessarily, $\mathcal{E}$ is of the form $(\sigma Z(f{:}F) = \varphi)\ \mathcal{F}$, where $\mathcal{F}$ is of length m. Let $X \notin \mathsf{bnd}((\sigma Z(f{:}F) = \varphi)\ \mathcal{F})$, and let η, ε be arbitrary environments. We derive:

$$\begin{array}{ll} & [\![(\sigma Z(f{:}F) = \varphi)\ \mathcal{F}]\!]\eta\varepsilon(X) \\ = & ([\![\mathcal{F}]\!]\eta[Z \mapsto (\sigma g \in \mathbb{B}^{\mathbb{F}}.\ \lambda v \in \mathbb{F}.\ [\![\varphi]\!]([\![\mathcal{F}]\!]\eta[Z \mapsto g]\varepsilon)\varepsilon[f \mapsto v])]\varepsilon)(X) \\ \stackrel{(IH)}{=} & (\eta[Z \mapsto (\sigma g \in \mathbb{B}^{\mathbb{F}}.\ \lambda v \in \mathbb{F}.\ [\![\varphi]\!]([\![\mathcal{F}]\!]\eta[Z \mapsto g]\varepsilon)\varepsilon[f \mapsto v])])(X) \\ \stackrel{X \neq Z}{=} & \eta(X). \qquad \square \end{array}$$

We finish this section with the following two observations:

1. The *equivalence checking problem*, for strong, branching and weak bisimilarity, can be encoded as the problem of solving a Parameterised Boolean Equation System, see [5]; the encoding takes two Linear Process Equations as input and yields a single Parameterised Boolean Equation System. The encodings are inspired by the early work by Lin [31].
2. The *local μ-calculus model checking problem* can be encoded as the problem of solving a Parameterised Boolean Equation System, see [1,3,32]; the encoding takes a Linear Process Equation and a μ-calculus formula and yields a single Parameterised Boolean Equation System.

3.2. Boolean Equation Systems

A special class of PBESs is the class of *Boolean Equation Systems* (BESs). BESs have been studied extensively in the literature [8]; due to the absence of data expressions and first order constructs, and due to the use of proposition variables rather than predicate variables, BESs are easier to understand, as the underlying lattice is simpler. Formally, we have:

Definition 9 (*Boolean Equation System*)**.** A *Boolean Equation System* is a PBES in which every predicate variable is of type B and every formula φ adheres to the following grammar (hereafter referred to as *proposition formulae*):

$$\varphi ::= \top \mid \bot \mid X \mid \varphi \oplus \varphi$$

where $\oplus \in \{\wedge, \vee\}$ and X is a proposition variable.

Mader [8] introduces *Infinite* Boolean Equation Systems *(IBESs)* as a vehicle for solving a model checking problem for infinite state systems. IBESs resemble BESs but differ in the following aspects: (1) finite *and* (countably) infinite conjunction

and disjunction over proposition variables are allowed, and (2) finite *and* (countably) infinite sequences of equations are allowed (but still only finitely many *blocks* of equations).

Definition 10 (*Infinite proposition formulae*)**.** *Infinite proposition formulae* ω are defined by the following grammar, for any countable sorts I and $J \subseteq I$:

$$\omega ::= \top | \bot | X_i \mid \omega \oplus \omega | \bigoplus_{j \in J} \omega$$

where $\oplus \in \{\wedge, \vee\}$, $\bigoplus \in \{\bigwedge, \bigvee\}$ and $X_i{:}B$ is a proposition variable for any $i \in I$.

Here, $\bigwedge_{j \in J}$ and $\bigvee_{j \in J}$ denote the infinite conjunction and disjunction over basic elements of a countable sort J, respectively.

Definition 11 (*Infinite Boolean Equation System*)**.** An *Infinite Boolean Equation System* (IBES) is inductively defined as follows:

- ϵ is the empty IBES;
- for every IBES $\mathcal{E}$, $\sigma\mathcal{B}\,\mathcal{E}$ is also an IBES, where $\sigma \in \{\mu, \nu\}$ is a fixpoint symbol and $\sigma\mathcal{B}$ is a block of equations $\{\sigma X_j = \omega_j | j \in J\}$ where J is a countable sort, and for each $j \in J$, $X_j{:}B$ is a proposition variable and ω_j is an infinite proposition formula.

Notice that BESs are, syntactically, exactly in the intersection of PBESs and IBESs. The notions of binding and occurring variables, and the induced notions of *open, closed* and *well-formedness*, that are defined for PBESs naturally transfer to IBESs. We also restrict to IBESs that are well-formed.

The semantics of infinite proposition formulae is defined in the context of a proposition environment $\eta{:}\mathcal{X} \to \mathbb{B}$. For any countable sort I, environment η and function $f{:}I \to \mathbb{B}$ we denote by $\eta[X_I \mapsto f]$ the simultaneous substitution of $f(i)$ for X_i in η for all $i \in I$, such that $\eta[X_I \mapsto f](X_i) = f(i)$ if $i \in I$ and $\eta(X_i)$ otherwise.

Definition 12 (*Semantics of infinite proposition formulae*)**.** Let $\eta{:}\mathcal{X} \to \mathbb{B}$ be a proposition environment. The *interpretation* $[\![\omega]\!]\eta$ that maps an infinite proposition formula ω to $\top$ or $\bot$, is inductively defined as follows:

$$\begin{aligned}
&[\![\top]\!]\eta && \triangleq \top \\
&[\![\bot]\!]\eta && \triangleq \bot \\
&[\![X_i]\!]\eta && \triangleq \eta(X_i) \\
&[\![\omega_1 \oplus \omega_2]\!]\eta && \triangleq [\![\omega_1]\!]\eta \oplus [\![\omega_2]\!]\eta \\
&[\![\textstyle\bigoplus_{j \in J} \omega]\!]\eta && \triangleq \mathsf{Q} v \in J\,.\,[\![\omega[j := v]]\!]\eta
\end{aligned}$$

where $\mathsf{Q} = \forall$ if $\bigoplus = \bigwedge$, and $\mathsf{Q} = \exists$ otherwise.

The set of functions $f{:}I \to \mathbb{B}$, where I is some (countable) sort, is denoted by $\mathbb{B}^I$. Together with the ordering $\sqsubseteq$, the set $(\mathbb{B}^I, \sqsubseteq)$ is a complete lattice. Let $\Omega = \{\omega_i \mid i \in I\}$ be a countable set of infinite proposition formulae. The functional induced by the interpretation of Ω is written $(\lambda i \in I.\ [\![\omega_i]\!]\eta)$, with $\omega_i \in \Omega$. This leads to the following transformer on infinite proposition formulae:

$$\lambda g \in \mathbb{B}^I.\ (\lambda i \in I.\ [\![\omega_i]\!]\eta[X_I \mapsto g]).$$

The transformer is a monotone operator on the complete lattice $(\mathbb{B}^I, \sqsubseteq)$, guaranteeing the existence of its least and greatest fixpoints.

Definition 13 (*Solution of an Infinite BES*)**.** Let η be a proposition environment, $\mathcal{E}$ be an IBES and $\sigma\mathcal{B} = \{\sigma X_i = \omega_i | i \in I\}$ be a block for some countable sort I. The *solution of an IBES* is inductively defined as follows:

$$\begin{aligned}
&[\![\epsilon]\!]\eta && \triangleq \eta \\
&[\![\sigma\mathcal{B}\,\mathcal{E}]\!]\eta && \triangleq [\![\mathcal{E}]\!]\eta[X_I \mapsto \sigma f \in \mathbb{B}^I.\ \lambda i \in I.[\![\omega_i]\!]([\![\mathcal{E}]\!]\eta[X_I \mapsto f])].
\end{aligned}$$

The solution of an IBES assigns a value to *every* proposition variable that occurs as a binding variable in the IBES. Often, only the value for specific proposition variables is sought, *e.g.* in local model checking. In such a case, equations that are unimportant to the solution of that variable can be pruned, yielding a smaller IBES, or even a BES. This follows from the following results.

Lemma 2. *Let $\mathcal{F}$ and $\mathcal{G}$ be IBESs. Let η be an arbitrary environment. Then:*

$$\mathsf{occ}(\mathcal{G}) \cap \mathsf{bnd}(\mathcal{F}) = \emptyset \ \textit{implies}\ \forall X \notin \mathsf{bnd}(\mathcal{F})\,.\,[\![\mathcal{F}\,\mathcal{G}]\!]\eta(X) = [\![\mathcal{G}]\!]\eta(X).$$

Proof. We use induction on the number of blocks in $\mathcal{F}$.

- Base case: $\mathcal{F} = \epsilon$. Let η be an arbitrary environment. Then $[\![\mathcal{F}\ \mathcal{G}]\!]\eta(X) = [\![\epsilon\ \mathcal{G}]\!]\eta(X) = [\![\mathcal{G}]\!]\eta(X)$ for all proposition variables X.
- Inductive step: assume that for some IBES $\mathcal{F}'$ we have for all θ:

$$\mathsf{occ}(\mathcal{G}) \cap \mathsf{bnd}(\mathcal{F}') = \emptyset \text{ implies } \forall X \notin \mathsf{bnd}(\mathcal{F}) \,.\, [\![\mathcal{F}'\ \mathcal{G}]\!]\theta(X) = [\![\mathcal{G}]\!]\theta(X). \tag{IH}$$

Assume that $\mathsf{occ}(\mathcal{G}) \cap \mathsf{bnd}(\sigma\, \mathcal{B}\, \mathcal{F}') = \emptyset$. Let $X \notin \mathsf{bnd}(\sigma\, \mathcal{B}\, \mathcal{F}')$. Then:

$$\begin{aligned}
&[\![\sigma\, \mathcal{B}\, \mathcal{F}'\ \mathcal{G}]\!]\eta(X) \\
&= [\![\mathcal{F}'\ \mathcal{G}]\!]\eta[X_I \mapsto \sigma X_I.\, \mathcal{B}([\![\mathcal{F}'\ \mathcal{G}]\!]\eta)](X) \\
&= \{\, \mathsf{occ}(\mathcal{G}) \cap \mathsf{bnd}(\sigma\, \mathcal{B}\, \mathcal{F}') = \emptyset \text{ implies } \mathsf{occ}(\mathcal{G}) \cap \mathsf{bnd}(\mathcal{F}') = \emptyset,\ \text{apply (IH)} \,\} \\
&\quad [\![\mathcal{G}]\!]\eta[X_I \mapsto \sigma X_I.\, \mathcal{B}([\![\mathcal{F}'\ \mathcal{G}]\!]\eta)](X) \\
&= \{\, X_I \cap \mathsf{occ}(\mathcal{G}) = \emptyset, X \notin X_I \,\} \\
&\quad [\![\mathcal{G}]\!]\eta(X). \qquad \square
\end{aligned}$$

By the previous lemma, irrelevant equations at the start of an IBES can be removed. The following proposition generalises this result to equations that occur anywhere in the IBES.

Proposition 1. *Let $\mathcal{E}$, $\mathcal{F}$, $\mathcal{G}$ be arbitrary IBESs. Then for all environments η:*

$$\mathsf{occ}(\mathcal{E}\ \mathcal{G}) \cap \mathsf{bnd}(\mathcal{F}) = \emptyset \text{ implies } \forall X \notin \mathsf{bnd}(\mathcal{F}) \,.\, [\![\mathcal{E}\ \mathcal{F}\ \mathcal{G}]\!]\eta(X) = [\![\mathcal{E}\ \mathcal{G}]\!]\eta(X).$$

Proof. We use induction on the number of blocks in $\mathcal{E}$.

- Base case: $\mathcal{E}$ consists of zero blocks. Then $\mathsf{occ}(\mathcal{G}) \cap \mathsf{bnd}(\mathcal{F}) = \emptyset$ and by Lemma 2 we have $[\![\mathcal{F}\ \mathcal{G}]\!]\eta(X) = [\![\mathcal{G}]\!]\eta(X)$ for all $X \notin \mathsf{bnd}(\mathcal{F})$.
- Inductive step: assume that for an IBES $\mathcal{E}'$ and all environments θ we have:

$$\begin{aligned}
&\mathsf{occ}(\mathcal{E}'\ \mathcal{G}) \cap \mathsf{bnd}(\mathcal{F}) = \emptyset \text{ implies} \\
&\quad \forall X \notin \mathsf{bnd}(\mathcal{F}) \,.\, [\![\mathcal{E}'\ \mathcal{F}\ \mathcal{G}]\!]\eta(X) = [\![\mathcal{E}'\ \mathcal{G}]\!]\eta(X).
\end{aligned} \tag{IH}$$

Suppose that $\mathsf{occ}(\sigma\, \mathcal{B}\, \mathcal{E}'\ \mathcal{G}) \cap \mathsf{bnd}(\mathcal{F}) = \emptyset$. Let $X \notin \mathsf{bnd}(\mathcal{F})$. Then:

$$\begin{aligned}
&[\![\sigma\, \mathcal{B}\, \mathcal{E}'\ \mathcal{F}\ \mathcal{G}]\!]\eta(X) \\
&= \{\, \text{Definition of semantics} \,\} \\
&\quad [\![\mathcal{E}'\ \mathcal{F}\ \mathcal{G}]\!]\eta[X_I \mapsto \sigma X_I.\, \mathcal{B}([\![\mathcal{E}'\ \mathcal{F}\ \mathcal{G}]\!]\eta)](X) \\
&= \{\, \text{apply (IH)} \,\} \\
&\quad [\![\mathcal{E}'\ \mathcal{G}]\!]\eta[X_I \mapsto \sigma X_I.\, \mathcal{B}([\![\mathcal{E}'\ \mathcal{F}\ \mathcal{G}]\!]\eta)](X) \\
&= \{\, \mathsf{occ}(\sigma\, \mathcal{B}\, \mathcal{E}'\ \mathcal{G}) \cap \mathsf{bnd}(\mathcal{F}) = \emptyset \text{ implies } \mathcal{B}([\![\mathcal{E}'\ \mathcal{F}\ \mathcal{G}]\!]\eta) = \mathcal{B}([\![\mathcal{E}'\ \mathcal{G}]\!]\eta) \,\} \\
&\quad [\![\mathcal{E}'\ \mathcal{G}]\!]\eta[X_I \mapsto \sigma X_I.\, \mathcal{B}([\![\mathcal{E}'\ \mathcal{G}]\!]\eta)](X) \\
&= \{\, \text{Definition of semantics} \,\} \\
&\quad [\![\sigma\, \mathcal{B}\, \mathcal{E}'\ \mathcal{G}]\!]\eta(X). \qquad \square
\end{aligned}$$

Infinite BESs are used in Section 5 when we instantiate PBESs containing countably infinite domains. We first look at instantiation on finite domains, for which normal BESs suffice.

4. Instantiation on finite domains

Instantiation is a transformation on PBESs that, for every variable X from a given set $\mathcal{P}$, replaces the equation $(\sigma X(d{:}D, e{:}E) = \varphi)$ by an entire PBES $(\sigma X_{d_1}(e{:}E) = \varphi_{d_1}) \cdots (\sigma X_{d_n}(e{:}E) = \varphi_{d_n})$. The transformation is given as Algorithm 1 for general PBESs $\mathcal{E}$ and arbitrary sets $\mathcal{P}$. Although the basic idea of the transformation is elementary, the devil is in the detail: careful bookkeeping and a naming scheme have to be applied to make the transformation work. This is taken care of by the function $\mathsf{Sub}_{\mathcal{P}}$ that is used in the main transformation $\mathsf{Inst}_{\mathcal{P}}$. It correctly introduces new predicate variables in the

Algorithm 1 The instantiation algorithm $\mathsf{Inst}_{\mathcal{P}}$.

For any $\mathcal{P} \subseteq \mathcal{X}$, with $\mathcal{P} \neq \emptyset$:

$$\mathsf{Inst}_{\emptyset}(\mathcal{E}) \triangleq \mathcal{E}$$
$$\mathsf{Inst}_{\mathcal{P}}(\epsilon) \triangleq \epsilon$$
$$\mathsf{Inst}_{\mathcal{P}}((\sigma X(d{:}D,\, e{:}E) = \varphi)\ \mathcal{E}) \triangleq$$
$$\begin{cases} \{(\sigma X_v(e{:}E) = \mathsf{Sub}_{\mathcal{P}}(\varphi[d := v]))|v \in D\}\ \mathsf{Inst}_{\mathcal{P}}(\mathcal{E}) & \text{if } X \in \mathcal{P} \\ (\sigma X(d{:}D,\, e{:}E) = \mathsf{Sub}_{\mathcal{P}}(\varphi))\ \mathsf{Inst}_{\mathcal{P}}(\mathcal{E}) & \text{otherwise} \end{cases}$$

where

$$\mathsf{Sub}_{\emptyset}(\varphi) \triangleq \varphi$$
$$\mathsf{Sub}_{\mathcal{P}}(b) \triangleq b$$
$$\mathsf{Sub}_{\mathcal{P}}(X(d,e)) \triangleq \begin{cases} \bigvee_{v\in D}(v = d \wedge X_v(e)) & \text{if } X \in \mathcal{P} \\ X(d,e) & \text{otherwise} \end{cases}$$
$$\mathsf{Sub}_{\mathcal{P}}(\varphi_1 \oplus \varphi_2) \triangleq \mathsf{Sub}_{\mathcal{P}}(\varphi_1) \oplus \mathsf{Sub}_{\mathcal{P}}(\varphi_2)$$
$$\mathsf{Sub}_{\mathcal{P}}(\mathsf{Q}d{:}D.\ \varphi) \triangleq \mathsf{Q}d{:}D.\ \mathsf{Sub}_{\mathcal{P}}(\varphi)$$

right-hand sides of the equations of the PBES, as we prove below. In the definition of $\mathsf{Sub}_{\mathcal{P}}$, the operand $\bigvee_{v\in D}$ abbreviates a finite disjunction over all basic elements v in D.

The soundness of the transformation is far from obvious due to the newly introduced predicate variables and the modifications to the right-hand sides. We prove that the transformation indeed preserves the solution of the original PBES, and claim a precise correspondence between the original PBES and the transformed PBES. Since the main proof is involved, we first prove correctness of Algorithm 1 for $|\mathcal{P}| = 1$ – *i.e.* when a single variable is instantiated – in Section 4.1. The correctness proof for the general case (arbitrary $\mathcal{P}$) is given in Section 4.2 and relies on the results of Section 4.1.

Without loss of generality, we assume that all predicate variables in this section are either of type $D \times E \to B$ or of type $E \to B$, for some finite sort D and some possibly infinite sort E. The finite sort D is used as the sort that is instantiated for a given predicate variable. We use the sort F when we mean either domain D or $D \times E$. With each predicate variable $X : D \times E \to B$ we associate a finite set of predicate variables $\mathsf{all}(X) \triangleq \{X_d : E \to B | d \in D\}$. For any PBES $\mathcal{E}$, we say that the predicate variable X is *instantiation-fresh* for $\mathcal{E}$ iff $\mathsf{all}(X) \cap \mathsf{var}(\mathcal{E}) = \emptyset$.

4.1. Instantiation for a single predicate variable

In order to facilitate the proof of the main theorem of this section, we first address several lemmas concerning the functions $\mathsf{Sub}_{\{X\}}$ and $\mathsf{Inst}_{\{X\}}$ to which we refer as Sub_X and Inst_X for conciseness. The soundness of Sub_X is established by the following lemma: for any φ the interpretations of φ and $\mathsf{Sub}_X(\varphi)$ within the context of an environment η correspond, provided that $\eta(X)([\![v]\!]) = \eta(X_v)$ for all $v \in D$.

Lemma 3. *Let φ be a predicate formula and $X{:}D \times E \to B$ be a predicate variable. Let η be an environment such that $\eta(X)([\![v]\!]) = \eta(X_v)$ for all $v \in D$. Then for any environment ε, $[\![\varphi]\!]\eta\varepsilon = [\![\mathsf{Sub}_X(\varphi)]\!]\eta\varepsilon$.*

Proof. Let ε be a data environment. We prove the statement by induction on the structure of φ.

1. $\varphi \equiv b$. This holds trivially.
2. $\varphi \equiv X(d,e)$. Then, using isomorphism between $\mathbb{D} \times \mathbb{E} \to \mathbb{B}$ and $\mathbb{D} \to \mathbb{E} \to \mathbb{B}$:
$$\begin{aligned} [\![X(d,e)]\!]\eta\varepsilon &= \eta(X)([\![d]\!]\varepsilon)([\![e]\!]\varepsilon) \;=\; \bigvee_{v\in D}([\![v]\!] = [\![d]\!]\varepsilon \wedge \eta(X)([\![v]\!])([\![e]\!]\varepsilon)) \\ &= \bigvee_{v\in D}([\![v]\!] = [\![d]\!]\varepsilon \wedge \eta(X_v)([\![e]\!]\varepsilon)) \\ &= [\![\bigvee_{v\in D}(v = d \wedge X_v(e))]\!]\eta\varepsilon \;=\; [\![\mathsf{Sub}_X(X(d,e))]\!]\eta\varepsilon. \end{aligned}$$
3. $\varphi \equiv Y(d,e)$ for $Y \neq X$. This holds trivially.
4. We assume for formulae φ_i, where $i \in \{1,2\}$:
$$[\![\varphi_i]\!]\eta\varepsilon = [\![\mathsf{Sub}_X(\varphi_i)]\!]\eta\varepsilon. \qquad \text{(IH)}$$

(a) $\varphi \equiv \varphi_1 \oplus \varphi_2$. Then:

$$\begin{aligned}[\![\varphi_1 \oplus \varphi_2]\!]\eta\varepsilon &= [\![\varphi_1]\!]\eta\varepsilon \oplus [\![\varphi_2]\!]\eta\varepsilon \overset{(IH)}{=} [\![\mathsf{Sub}_X(\varphi_1)]\!]\eta\varepsilon \oplus [\![\mathsf{Sub}_X(\varphi_2)]\!]\eta\varepsilon \\ &= [\![\mathsf{Sub}_X(\varphi_1 \oplus \varphi_2)]\!]\eta\varepsilon.\end{aligned}$$

(b) $\varphi \equiv \mathsf{Q}f{:}F\,.\,\varphi_1$. Then:

$$\begin{aligned}[\![\mathsf{Q}f{:}F\,.\,\varphi_1]\!]\eta\varepsilon &= \mathsf{Q}w \in \mathbb{F}\,.\,[\![\varphi_1]\!]\eta(\varepsilon[f \mapsto w]) \\ &\overset{(IH)}{=} \mathsf{Q}w \in \mathbb{F}\,.\,[\![\mathsf{Sub}_X(\varphi_1)]\!]\eta(\varepsilon[f \mapsto w]) \\ &= [\![\mathsf{Sub}_X(\mathsf{Q}f{:}F\,.\,\varphi_1)]\!]\eta\varepsilon. \qquad \square\end{aligned}$$

In the correctness proof below, we will encounter PBESs in which an unbound predicate variable is instantiated. As shown by the following lemma, instantiating an unbound variable X in a PBES $\mathcal{E}$ does not change the solution of $\mathcal{E}$ in the context of an environment η, provided that $\eta(X)([\![v]\!]) = \eta(X_v)$ for all $v \in D$.

Lemma 4. *Let $\mathcal{E}$ be a PBES for which the predicate variable $X{:}D \times E \to B$ is instantiation-fresh and $X \notin \mathsf{bnd}(\mathcal{E})$. Let η be an environment such that $\eta(X)([\![v]\!]) = \eta(X_v)$ for all $v \in D$. Then for any environment ε, $[\![\mathcal{E}]\!]\eta\varepsilon = [\![\mathsf{Inst}_X(\mathcal{E})]\!]\eta\varepsilon$.*

Proof. Let ε be a data environment. We prove the lemma by induction on the length of $\mathcal{E}$. If $\mathcal{E}$ is of length 0 then: $[\![\epsilon]\!]\eta\varepsilon = \eta = [\![\mathsf{Inst}_X(\epsilon)]\!]\eta\varepsilon$. Suppose $\mathcal{E}$ is of length $m+1$ and for all $\mathcal{E}'$ of length m, we have, for any environment υ:

$$[\![\mathcal{E}']\!]\eta\upsilon = [\![\mathsf{Inst}_X(\mathcal{E}')]\!]\eta\upsilon. \tag{IH}$$

Necessarily, $\mathcal{E}$ is of the form $(\sigma Z(f{:}F) = \varphi)\ \mathcal{F}$, where $\mathcal{F}$ is of length m. We derive:

$$\begin{aligned}&[\![\mathcal{E}]\!]\eta\varepsilon = [\![(\sigma Z(f{:}F) = \varphi)\ \mathcal{F}]\!]\eta\varepsilon \\ &= [\![\mathcal{F}]\!]\eta[Z \mapsto (\sigma g \in \mathbb{B}^{\mathbb{F}}\,.\,\lambda v \in \mathbb{F}\,.\,[\![\varphi]\!]([\![\mathcal{F}]\!]\eta[Z \mapsto g]\varepsilon)\varepsilon[f \mapsto v])]\varepsilon \\ &\overset{*}{=} [\![\mathcal{F}]\!]\eta[Z \mapsto (\sigma g \in \mathbb{B}^{\mathbb{F}}\,.\,\lambda v \in \mathbb{F}\,.\,[\![\mathsf{Sub}_X(\varphi)]\!]([\![\mathcal{F}]\!]\eta[Z \mapsto g]\varepsilon)\varepsilon[f \mapsto v])]\varepsilon \\ &\overset{(IH)}{=} [\![\mathsf{Inst}_X(\mathcal{F})]\!]\eta[Z \mapsto (\sigma g \in \mathbb{B}^{\mathbb{F}}\,.\,\lambda v \in \mathbb{F}\,. \\ &\qquad [\![\mathsf{Sub}_X(\varphi)]\!]([\![\mathsf{Inst}_X(\mathcal{F})]\!]\eta[Z \mapsto g]\varepsilon)\varepsilon[f \mapsto v])]\varepsilon \\ &= [\![\mathsf{Inst}_X((\sigma Z(f{:}F) = \varphi)\ \mathcal{F})]\!]\eta\varepsilon \\ &= [\![\mathsf{Inst}_X(\mathcal{E})]\!]\eta\varepsilon\end{aligned}$$

where at $*$ we used the following equivalence:

$$(\sigma g \in \mathbb{B}^{\mathbb{F}}.[\![\varphi]\!]([\![\mathcal{F}]\!]\eta[Z \mapsto g]\varepsilon)\varepsilon) = (\sigma g \in \mathbb{B}^{\mathbb{F}}.[\![\mathsf{Sub}_X(\varphi)]\!]([\![\mathcal{F}]\!]\eta[Z \mapsto g]\varepsilon)\varepsilon)$$

which follows readily from Lemma 3. Observe that this lemma applies because $([\![\mathcal{F}]\!]\eta[Z \mapsto g]\varepsilon)(X)([\![v]\!]) = ([\![\mathcal{F}]\!]\eta[Z \mapsto g]\varepsilon)(X_v)$ for all $v \in D$ by assumption on η, instantiation-freshness of X for $\mathcal{E}$, $X \notin \mathsf{bnd}(\mathcal{F})$ and Lemma 1. $\square$

Suppose we have a PBES $\mathcal{E}$ in which the first equation is for variable X and X is instantiated in that PBES, yielding the PBES $\mathsf{Inst}_X(\mathcal{E})$. The following lemma states that the solution to X in the original PBES and the solutions to its instantiated counterparts in the resulting PBES correspond.

Lemma 5. *Let $\mathcal{F}$ be a PBES of the form $(\sigma X(d{:}D, e{:}E) = \varphi)\ \mathcal{E}$ such that X is instantiation-fresh for $\mathcal{F}$. Then for any environments η, ε:*

$$\forall v \in D\,.\,([\![\mathsf{Inst}_X(\mathcal{F})]\!]\eta\varepsilon)(X_v) = (([\![\mathcal{F}]\!]\eta\varepsilon)(X))([\![v]\!]).$$

Proof. Assume that $D = \{v_1, \ldots, v_n\}$; then $|D| = |\mathbb{D}| = n$ and take $i \in \{1, \ldots, n\}$. Let η, ε be environments and $\mathcal{E}_\iota \overset{\Delta}{=} \mathsf{Inst}_X(\mathcal{E})$. First, we rewrite the left-hand side of the equality as follows:

$$\begin{aligned}&([\![\mathsf{Inst}_X(\mathcal{F})]\!]\eta\varepsilon)(X_{v_i}) \\ &\overset{*}{=} \pi_i\big(\sigma(g_{v_1}, \ldots, g_{v_n}) \in (\mathbb{B}^{\mathbb{E}})^n\,. \\ &\quad (\lambda w \in \mathbb{E}\,.\,[\![\mathsf{Sub}_X(\varphi[d := v_1])]\!]([\![\mathcal{E}_\iota]\!]\eta[(X_{v_1}, \ldots, X_{v_n}) \mapsto (g_{v_1}, \ldots, g_{v_n})]\varepsilon)\varepsilon[e \mapsto w], \ldots, \\ &\quad \lambda w{\in}\mathbb{E}\,.\,[\![\mathsf{Sub}_X(\varphi[d := v_n])]\!]([\![\mathcal{E}_\iota]\!]\eta[(X_{v_1}, \ldots, X_{v_n}) \mapsto (g_{v_1}, \ldots, g_{v_n})]\varepsilon)\varepsilon[e \mapsto w])\big)\end{aligned}$$

$$\stackrel{\dagger}{=} (\sigma g \in \mathbb{B}^{\mathbb{D}\to\mathbb{E}} . (\lambda u \in \mathbb{D} . \lambda w \in \mathbb{E} . \\ [\![\mathsf{Sub}_X(\varphi)]\!]([\![\mathcal{E}_\iota]\!]\eta[X_{v_1} \mapsto g([\![v_1]\!]), \ldots, X_{v_n} \mapsto g([\![v_n]\!])]\varepsilon)\varepsilon[d \mapsto u, e \mapsto w]))([\![v_i]\!]).$$

At $*$ we used Bekič's theorem [33] to replace n nested σ-fixpoints by a simultaneous σ-fixpoint over an n-tuple, and the fact that $X_{v_i} \in \mathsf{bnd}(\mathsf{Inst}_X(\mathcal{F}))$. At $\dagger$ we used the assumption that the data theory is fully abstract, and the isomorphism between $(\mathbb{B}^{\mathbb{E}})^{|\mathbb{D}|}$ and $\mathbb{B}^{\mathbb{D}\to\mathbb{E}}$ to replace a tuple of functions $(g_{v_1}, \ldots, g_{v_n}) : (\mathbb{E} \to \mathbb{B})^n$ by a single function $g : \mathbb{D} \to \mathbb{E} \to \mathbb{B}$ such that for any $u \in D$: $g([\![u]\!]) = g_u$. For the right-hand side, we derive:

$$\begin{aligned}
&([\![\mathcal{F}]\!]\eta\varepsilon)(X([\![v_i]\!]\varepsilon)) \\
&\quad = (\sigma f \in \mathbb{B}^{\mathbb{D}\times\mathbb{E}} . \lambda(u, w) \in (\mathbb{D} \times \mathbb{E}) . [\![\varphi]\!]([\![\mathcal{E}]\!]\eta[X \mapsto f]\varepsilon)\varepsilon[(d, e) \mapsto (u, w)])([\![v_i]\!]) \\
&\quad = (\sigma f \in \mathbb{B}^{\mathbb{D}\to\mathbb{E}} . \lambda u \in \mathbb{D} . \lambda w \in \mathbb{E} . [\![\varphi]\!]([\![\mathcal{E}]\!]\eta[X \mapsto f]\varepsilon)\varepsilon[d \mapsto u, e \mapsto w])([\![v_i]\!]).
\end{aligned}$$

So it suffices to show the following equivalence:

$$\begin{aligned}
&(\sigma f \in \mathbb{B}^{\mathbb{D}\to\mathbb{E}} . \lambda u \in \mathbb{D} . \lambda w \in \mathbb{E} . [\![\varphi]\!]([\![\mathcal{E}]\!]\eta[X \mapsto f]\varepsilon)\varepsilon[d \mapsto u, e \mapsto w]) \\
&= (\sigma g \in \mathbb{B}^{\mathbb{D}\to\mathbb{E}} . (\lambda u \in \mathbb{D} . \lambda w \in \mathbb{E} . \\
&\qquad [\![\mathsf{Sub}_X(\varphi)]\!]([\![\mathcal{E}_\iota]\!]\eta[X_{v_1} \mapsto g([\![v_1]\!]), \ldots, X_{v_n} \mapsto g([\![v_n]\!])]\varepsilon)\varepsilon[d \mapsto u, e \mapsto w]))
\end{aligned}$$

which follows readily from:

$$\begin{aligned}
&[\![\varphi]\!]([\![\mathcal{E}]\!]\eta[X \mapsto h]\varepsilon)\upsilon \; = \\
&\quad [\![\mathsf{Sub}_X(\varphi)]\!]([\![\mathcal{E}_\iota]\!]\eta[X_{v_1} \mapsto h([\![v_1]\!]), \ldots, X_{v_n} \mapsto h([\![v_n]\!])]\varepsilon)\upsilon
\end{aligned} \tag{1}$$

for all environments υ and $h \in \mathbb{B}^{\mathbb{D}\to\mathbb{E}}$. Let υ be an environment and $h \in \mathbb{B}^{\mathbb{D}\to\mathbb{E}}$. We prove (1) using Lemmas 3 and 4 as follows:

$$\begin{aligned}
&[\![\varphi]\!]([\![\mathcal{E}]\!]\eta[X \mapsto h]\varepsilon)\upsilon \\
&\quad \stackrel{*}{=} [\![\varphi]\!]([\![\mathcal{E}]\!]\eta[X \mapsto h][X_{v_1} \mapsto h([\![v_1]\!]), \ldots, X_{v_n} \mapsto h([\![v_n]\!])]\varepsilon)\upsilon \\
&\quad \stackrel{3}{=} [\![\mathsf{Sub}_X(\varphi)]\!]([\![\mathcal{E}]\!]\eta[X \mapsto h][X_{v_1} \mapsto h([\![v_1]\!]), \ldots, X_{v_n} \mapsto h([\![v_n]\!])]\varepsilon)\upsilon \\
&\quad \stackrel{4}{=} [\![\mathsf{Sub}_X(\varphi)]\!]([\![\mathcal{E}_\iota]\!]\eta[X \mapsto h][X_{v_1} \mapsto h([\![v_1]\!]), \ldots, X_{v_n} \mapsto h([\![v_n]\!])]\varepsilon)\upsilon \\
&\quad \stackrel{\dagger}{=} [\![\mathsf{Sub}_X(\varphi)]\!]([\![\mathcal{E}_\iota]\!]\eta[X_{v_1} \mapsto h([\![v_1]\!]), \ldots, X_{v_n} \mapsto h([\![v_n]\!])]\varepsilon)\upsilon
\end{aligned}$$

where at $*$ we used that X is instantiation-fresh for $\mathcal{F}$ and at $\dagger$ we used that $X \notin \mathsf{occ}(\mathsf{Sub}_X(\varphi))$. □

Note that the previous lemma does not state that instantiation does not have undesirable side-effects. This topic is addressed by the following lemma. It establishes that, when instantiating the variable of the first equation of a PBES, the solutions for non-instantiated variables are unaffected.

Lemma 6. *Let $\mathcal{F} \stackrel{\Delta}{=} (\sigma X(d{:}D, e{:}E) = \varphi)\ \mathcal{E}$ be a PBES and let X be instantiation-fresh for $\mathcal{F}$. Then for all environments η, ε:*

$$\forall Y \in \mathcal{X} . Y \notin \mathsf{all}(X) \cup \{X\} \implies ([\![\mathsf{Inst}_X(\mathcal{F})]\!]\eta\varepsilon)(Y) = ([\![\mathcal{F}]\!]\eta\varepsilon)(Y).$$

Proof. Let η, ε be environments and $Y \in \mathcal{X}$ such that $Y \notin \mathsf{all}(X) \cup \{X\}$. Let $g : \mathbb{D} \times \mathbb{E} \to \mathbb{B}$ be such that:

$$\forall v \in D . g([\![v]\!]) = ([\![\mathsf{Inst}_X(\mathcal{F})]\!]\eta\varepsilon)(X_v).$$

Then by Lemma 5, we have $g = ([\![\mathcal{F}]\!]\eta\varepsilon)(X)$ and, using Lemma 4:

$$\begin{aligned}
&([\![\mathsf{Inst}_X(\mathcal{F})]\!]\eta\varepsilon)(Y) \\
&\quad = ([\![\mathsf{Inst}_X(\mathcal{E})]\!]\eta[X_{v_1} \mapsto g([\![v_1]\!]), \ldots, X_{v_n} \mapsto g([\![v_n]\!])]\varepsilon)(Y) \\
&\quad = ([\![\mathsf{Inst}_X(\mathcal{E})]\!]\eta[X \mapsto g][X_{v_1} \mapsto g([\![v_1]\!]), \ldots, X_{v_n} \mapsto g([\![v_n]\!])]\varepsilon)(Y) \\
&\quad \stackrel{4}{=} ([\![\mathcal{E}]\!]\eta[X \mapsto g][X_{v_1} \mapsto g([\![v_1]\!]), \ldots, X_{v_n} \mapsto g([\![v_n]\!])]\varepsilon)(Y) \\
&\quad = ([\![\mathcal{E}]\!]\eta[X \mapsto g]\varepsilon)(Y) \\
&\quad = ([\![\mathcal{F}]\!]\eta\varepsilon)(Y). \qquad \square
\end{aligned}$$

We are now ready to prove the main theorem of this section, which generalises Lemmas 5 and 6: instantiation of a single, binding predicate variable X is sound for *arbitrary* PBESs, not just for PBESs in which X is bound in the first equation.

Theorem 1. *Let $\mathcal{E}$ be a PBES and $X \in \mathsf{bnd}(\mathcal{E})$ be instantiation-fresh for $\mathcal{E}$. Then for all environments η, ε:*

(a) $\forall v \in D \,.\, (\llbracket \mathsf{Inst}_X(\mathcal{E}) \rrbracket \eta\varepsilon)(X_v) = (\llbracket \mathcal{E} \rrbracket \eta\varepsilon)(X(\llbracket v \rrbracket))$

(b) $\forall Y \in \mathcal{X} \,.\, Y \notin \mathsf{all}(X) \cup \{X\} \implies (\llbracket \mathsf{Inst}_X(\mathcal{E}) \rrbracket \eta\varepsilon)(Y) = (\llbracket \mathcal{E} \rrbracket \eta\varepsilon)(Y)$.

Proof. Observe that $\mathcal{E}$ is of the following form:

$$\mathcal{E} \triangleq \mathcal{E}_1\, \mathcal{F} \quad \text{with} \quad \mathcal{F} \triangleq (\sigma X(d{:}D, e{:}E) = \varphi)\, \mathcal{E}_2$$

for some predicate formula φ and PBESs $\mathcal{E}_1$ and $\mathcal{E}_2$. We prove the claim by induction on the structure of $\mathcal{E}_1$. For $\mathcal{E}_1 = \epsilon$, statements (a) and (b) follow immediately due to Lemmas 5 and 6, respectively. Suppose $\mathcal{E}_1 = (\sigma' Z(f{:}F) = \psi)\, \mathcal{E}'$ for some PBES $\mathcal{E}'$. We assume as induction hypotheses, for all environments η', ε':

(IHa) $\forall v \in D \,.\, (\llbracket \mathsf{Inst}_X(\mathcal{E}'\, \mathcal{F}) \rrbracket \eta'\varepsilon')(X_v) = (\llbracket \mathcal{E}'\, \mathcal{F} \rrbracket \eta'\varepsilon')(X(\llbracket v \rrbracket))$

(IHb) $\forall Y \in \mathcal{X} \,.\, Y \notin \mathsf{all}(X) \cup \{X\} \implies (\llbracket \mathsf{Inst}_X(\mathcal{E}'\, \mathcal{F}) \rrbracket \eta'\varepsilon')(Y) = (\llbracket \mathcal{E}'\, \mathcal{F} \rrbracket \eta'\varepsilon')(Y)$.

Let $v \in D$. Then for statement (a) we derive:

$$\begin{aligned}
&(\llbracket \mathsf{Inst}_X((\sigma' Z(f{:}F) = \psi)\, \mathcal{E}'\, \mathcal{F}) \rrbracket \eta\varepsilon)(X_v) \\
&\quad = (\llbracket (\sigma' Z(f{:}F) = \mathsf{Sub}_X(\psi))\, \mathsf{Inst}_X(\mathcal{E}'\, \mathcal{F}) \rrbracket \eta\varepsilon)(X_v) \\
&\quad = (\llbracket \mathsf{Inst}_X(\mathcal{E}'\, \mathcal{F}) \rrbracket \eta[Z \mapsto (\sigma' h \in \mathbb{B}^{\mathbb{F}} \,. \\
&\qquad\qquad \lambda u \in \mathbb{F} \,.\, \llbracket \mathsf{Sub}_X(\psi) \rrbracket (\llbracket \mathsf{Inst}_X(\mathcal{E}'\, \mathcal{F}) \rrbracket \eta[Z \mapsto h]\varepsilon))]\varepsilon[f \mapsto u])(X_v) \\
&\quad \overset{*}{=} (\llbracket \mathcal{E}'\, \mathcal{F} \rrbracket \eta[Z \mapsto \sigma' h \in \mathbb{B}^{\mathbb{F}} \,.\, \lambda u \in \mathbb{F} \,.\, \llbracket \psi \rrbracket (\llbracket \mathcal{E}'\, \mathcal{F} \rrbracket \eta[Z \mapsto h]\varepsilon)\varepsilon[f \mapsto u]]\varepsilon)(X(\llbracket v \rrbracket)) \\
&\quad = (\llbracket (\sigma' Z(f{:}F) = \psi)\, \mathcal{E}'\, \mathcal{F} \rrbracket \eta\varepsilon)(X(\llbracket v \rrbracket)).
\end{aligned}$$

At $*$ we used (IHa) and Lemma 3 using both (IHa) and (IHb). The derivation for statement (b) follows the same line of reasoning and is therefore omitted. □

We demonstrate the use of Inst_X by applying it to an example.

Example 4. Consider the following PBES $\mathcal{E}$:

$$\begin{aligned}
\nu X(b{:}B) &= \exists n{:}N \,.\, Y(n) \wedge b \\
\mu Y(n{:}N) &= X(n \geq 10).
\end{aligned}$$

Instantiation of parameter b of X yields the PBES $\mathcal{E}'$ below, after minor rewriting:

$$\begin{aligned}
\nu X_\top &= \exists n{:}N \,.\, Y(n) \\
\nu X_\bot &= \bot \\
\mu Y(n{:}N) &= (n \geq 10 \wedge X_\top) \vee (n < 10 \wedge X_\bot).
\end{aligned}$$

The PBES $\mathcal{E}'$ can be solved using migration, substitution and subsequent logic rewriting, which yields:

$$\begin{aligned}
\nu X_\top &= \top \\
\mu Y(n{:}N) &= n \geq 10 \\
\nu X_\bot &= \bot.
\end{aligned}$$

Hence, for arbitrary environments η, ε, we have the following correspondences:

- $(\llbracket \mathcal{E} \rrbracket \eta\varepsilon)(X)(\top) = (\llbracket \mathcal{E}' \rrbracket \eta\varepsilon)(X_\top) = \top$,
- $(\llbracket \mathcal{E} \rrbracket \eta\varepsilon)(X)(\bot) = (\llbracket \mathcal{E}' \rrbracket \eta\varepsilon)(X_\bot) = \bot$,
- $(\llbracket \mathcal{E} \rrbracket \eta\varepsilon)(Y) = (\llbracket \mathcal{E}' \rrbracket \eta\varepsilon)(Y) = (\lambda n{:}\mathbb{N} \,.\, n \geq 10)$.

Instantiation allows for solving a rather complex PBES $\mathcal{E}$ using standard PBES manipulation techniques and instantiation of a single predicate variable.

By Theorem 1 we have established the correctness of the instantiation algorithm $\mathsf{Inst}_{\mathcal{P}}$ when $\mathcal{P} = \{X\}$ for some variable X. We now consider the more general case where an arbitrary set of variables $\mathcal{P}$ is instantiated simultaneously.

4.2. Simultaneous instantiation

Instantiation of a set of variables $\mathcal{P}$ in a PBES can be achieved by successively applying Inst_X for every $X \in \mathcal{P}$. Sound as this strategy may be, it is undesirable as it is highly inefficient. The simultaneous instantiation performed by $\mathsf{Inst}_{\mathcal{P}}$ is

more efficient because it requires only a single pass over the entire PBES. We now prove soundness of $\mathsf{Inst}_{\mathcal{P}}$ for any set $\mathcal{P}$ by showing that applying $\mathsf{Inst}_{\mathcal{P}}$ yields a PBES that is syntactically equivalent to the one obtained by successively applying Inst_X for every $X \in \mathcal{P}$. First, we prove several lemmas concerning $\mathsf{Inst}_{\mathcal{P}}$ and $\mathsf{Sub}_{\mathcal{P}}$. For a given formula φ, set of variables $\mathcal{P}$ and $X \in \mathcal{P}$, the following lemma states that applying $\mathsf{Sub}_{\mathcal{P}}$ to φ yields the same result as applying Sub_X after $\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}$ to φ.

Lemma 7. *Let φ be a predicate formula and $\mathcal{P}$ be a non-empty set of predicate variables such that for all $X, Y \in \mathcal{P}$, $X \notin \mathsf{all}(Y)$. Then for all $X \in \mathcal{P}$:*

$$\mathsf{Sub}_{\mathcal{P}}(\varphi) = \mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi)).$$

Proof. Let $X \in \mathcal{P}$. We prove the lemma by structural induction on φ.

1. $\varphi \equiv b$. Trivial.
2. $\varphi \equiv X(d, e)$. Then:

$$\begin{aligned}\mathsf{Sub}_{\mathcal{P}}(X(d,e)) &= \textstyle\bigvee_{v\in D}(v = d \wedge X_v(e)) \;=\; \mathsf{Sub}_X(X(d,e)) \\ &= \mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(X(d,e))).\end{aligned}$$

3. $\varphi \equiv Y(d, e)$ for some $Y \in \mathcal{X}$ with $Y \neq X$. If $Y \notin \mathcal{P}$ then this case is trivial. If $Y \in \mathcal{P}$ then:

$$\begin{aligned}\mathsf{Sub}_{\mathcal{P}}(Y(d,e)) &= \textstyle\bigvee_{v\in D}(v = d \wedge Y_v(e)) \overset{*}{=} \mathsf{Sub}_X(\bigvee_{v\in D}(v = d \wedge Y_v(e))) \\ &\overset{\dagger}{=} \mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(Y(d,e)))\end{aligned}$$

   where at $*$ we used $X \notin \mathsf{all}(Y)$ and at $\dagger$ we used $Y \in \mathcal{P}\setminus\{X\}$.
4. Assume for predicate formulae φ_i, $i \in \{1, 2\}$:

$$\mathsf{Sub}_{\mathcal{P}}(\varphi_i) = \mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi_i)). \tag{IH}$$

   (a) $\varphi \equiv \varphi_1 \oplus \varphi_2$ for some $\oplus \in \{\vee, \wedge\}$. Then:

$$\begin{aligned}\mathsf{Sub}_{\mathcal{P}}(\varphi_1 \oplus \varphi_2) &= \mathsf{Sub}_{\mathcal{P}}(\varphi_1) \oplus \mathsf{Sub}_{\mathcal{P}}(\varphi_2) \\ &\overset{(IH)}{=} \mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi_1)) \oplus \mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi_2)) \\ &= \mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi_1 \oplus \varphi_2)).\end{aligned}$$

   (b) $\varphi \equiv \mathsf{Q}d{:}D.\varphi_1$ for some $\mathsf{Q} \in \{\exists, \forall\}$. Then:

$$\begin{aligned}\mathsf{Sub}_{\mathcal{P}}(\mathsf{Q}d{:}D.\,\varphi_1) &= \mathsf{Q}d{:}D.\,\mathsf{Sub}_{\mathcal{P}}(\varphi_1) \\ &\overset{(IH)}{=} \mathsf{Q}d{:}D.\,\mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi_1)) \\ &= \mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\mathsf{Q}d{:}D\,.\,\varphi_1)).\end{aligned}$$ □

The next lemma establishes a similar result for $\mathsf{Inst}_{\mathcal{P}}$ instead of $\mathsf{Sub}_{\mathcal{P}}$.

Lemma 8. *Let $\mathcal{E}$ be a PBES and $\mathcal{P} \subseteq \mathsf{bnd}(\mathcal{E})$ be a non-empty set of predicate variables that is instantiation-fresh for $\mathcal{E}$. Then for all $X \in \mathcal{P}$:*

$$\mathsf{Inst}_{\mathcal{P}}(\mathcal{E}) = \mathsf{Inst}_X(\mathsf{Inst}_{\mathcal{P}\setminus\{X\}}(\mathcal{E})).$$

Proof. In this proof we rely on Lemma 7. This lemma applies because from $\mathcal{P} \subseteq \mathsf{bnd}(\mathcal{E})$ and the fact that $\mathcal{P}$ is instantiation-fresh for $\mathcal{E}$, it follows that for all $X, Y \in \mathcal{P}$, $X \notin \mathsf{all}(Y)$. We prove the lemma by induction on the length of $\mathcal{E}$. The case $\mathcal{E} = \epsilon$ is trivial. Suppose $\mathcal{E} = (\sigma Y(d{:}D, e{:}E) = \varphi)\ \mathcal{E}'$ for some PBES $\mathcal{E}'$. Assume that for all $X \in \mathcal{P}$:

$$\mathsf{Inst}_{\mathcal{P}}(\mathcal{E}') = \mathsf{Inst}_X(\mathsf{Inst}_{\mathcal{P}\setminus\{X\}}(\mathcal{E}')). \tag{IH}$$

Let $X \in \mathcal{P}$. We distinguish three cases and use Lemma 7 and (IH) at every $*$:

1. $Y \notin \mathcal{P}$. Then:

$$\begin{aligned}&\mathsf{Inst}_{\mathcal{P}}((\sigma Y(d{:}D, e{:}E) = \varphi)\ \mathcal{E}') \\ &= (\sigma Y(d{:}D, e{:}E) = \mathsf{Sub}_{\mathcal{P}}(\varphi))\ \mathsf{Inst}_{\mathcal{P}}(\mathcal{E}') \\ &\overset{*}{=} (\sigma Y(d{:}D, e{:}E) = \mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi)))\ \mathsf{Inst}_X(\mathsf{Inst}_{\mathcal{P}\setminus\{X\}}(\mathcal{E}'))\end{aligned}$$

$$= \mathsf{Inst}_X((\sigma Y(d{:}D, e{:}E) = \mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi))\ \mathsf{Inst}_{\mathcal{P}\setminus\{X\}}(\mathcal{E}'))$$
$$= \mathsf{Inst}_X(\mathsf{Inst}_{\mathcal{P}\setminus\{X\}}((\sigma Y(d{:}D, e{:}E) = \varphi)\ \mathcal{E}')).$$

2. $Y \in \mathcal{P} \wedge Y \neq X$. Observe that $X \notin \mathsf{all}(Y)$. Then:

$$\begin{aligned}
&\mathsf{Inst}_{\mathcal{P}}((\sigma Y(d{:}D, e{:}E) = \varphi)\ \mathcal{E}') \\
&= \{(\sigma Y_v(e{:}E) = \mathsf{Sub}_{\mathcal{P}}(\varphi[d := v]))|v \in D\}\ \mathsf{Inst}_{\mathcal{P}}(\mathcal{E}') \\
&\overset{*}{=} \{(\sigma Y_v(e{:}E) = \mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi[d := v])))|v \in D\} \\
&\quad\ \mathsf{Inst}_X(\mathsf{Inst}_{\mathcal{P}\setminus\{X\}}(\mathcal{E}')) \\
&= \mathsf{Inst}_X(\{(\sigma Y_v(e{:}E) = \mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi[d := v]))|v \in D\}\ \mathsf{Inst}_{\mathcal{P}\setminus\{X\}}(\mathcal{E}')) \\
&= \mathsf{Inst}_X(\mathsf{Inst}_{\mathcal{P}\setminus\{X\}}((\sigma Y(d{:}D, e{:}E) = \varphi)\ \mathcal{E}')).
\end{aligned}$$

3. $Y = X$. Then:

$$\begin{aligned}
&\mathsf{Inst}_{\mathcal{P}}((\sigma X(d{:}D, e{:}E) = \varphi)\ \mathcal{E}') \\
&= \{(\sigma X_v(e{:}E) = \mathsf{Sub}_{\mathcal{P}}(\varphi[d := v]))|v \in D\}\ \mathsf{Inst}_{\mathcal{P}}(\mathcal{E}') \\
&\overset{*}{=} \{(\sigma X_v(e{:}E) = \mathsf{Sub}_X(\mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi[d := v])))|v \in D\} \\
&\quad\ \mathsf{Inst}_X(\mathsf{Inst}_{\mathcal{P}\setminus\{X\}}(\mathcal{E}')) \\
&= \mathsf{Inst}_X((\sigma X(d{:}D, e{:}E) = \mathsf{Sub}_{\mathcal{P}\setminus\{X\}}(\varphi))\ \mathsf{Inst}_{\mathcal{P}\setminus\{X\}}(\mathcal{E}')) \\
&= \mathsf{Inst}_X(\mathsf{Inst}_{\mathcal{P}\setminus\{X\}}((\sigma X(d{:}D, e{:}E) = \varphi)\ \mathcal{E}')). \qquad \square
\end{aligned}$$

We introduce the following shorthand notation for functional composition of Inst functions over a set of variables $\mathcal{P}$. Let $\leq$ be a total order on $\mathcal{X}$, being the set of all predicate variables. Then:

$$\bigcirc_{X\in\mathcal{P}}\mathsf{Inst}_X = \begin{cases} \mathcal{I} & \text{if } \mathcal{P} = \emptyset \\ \mathsf{Inst}_Y \circ \bigcirc_{X\in\mathcal{P}\setminus\{Y\}}\mathsf{Inst}_X & \text{for the } \leq\text{-minimal } Y \in \mathcal{P} \\ & \text{otherwise} \end{cases}$$

where $\mathcal{I}$ denotes the identity function for PBESs, *i.e.* $\mathcal{I}(\mathcal{E}) = \mathcal{E}$ for all $\mathcal{E}$. We now prove that instantiating a set of variables $\mathcal{P}$ yields the same equation system as successively instantiating for every variable in $\mathcal{P}$.

Lemma 9. *Let $\mathcal{E}$ be a PBES and $\mathcal{P} \subseteq \mathsf{bnd}(\mathcal{E})$ be a set of predicate variables that is instantiation-fresh for $\mathcal{E}$. Then:*

$$\mathsf{Inst}_{\mathcal{P}}(\mathcal{E}) = (\bigcirc_{X\in\mathcal{P}}\mathsf{Inst}_X)(\mathcal{E}).$$

Proof. The proof goes by induction on the size of $\mathcal{P}$. If $\mathcal{P} = \emptyset$ then trivially: $\mathsf{Inst}_{\mathcal{P}}(\mathcal{E}) = \mathcal{E} = \mathcal{I}(\mathcal{E}) = (\bigcirc_{X\in\mathcal{P}}\mathsf{Inst}_X)(\mathcal{E})$. Otherwise, let Y be the $\leq$-minimal element of $\mathcal{P}$ and assume:

$$\mathsf{Inst}_{\mathcal{P}\setminus\{Y\}}(\mathcal{E}) = (\bigcirc_{X\in\mathcal{P}\setminus\{Y\}}\mathsf{Inst}_X)(\mathcal{E}). \tag{IH}$$

Then, using Lemma 8:

$$\begin{aligned}
\mathsf{Inst}_{\mathcal{P}}(\mathcal{E}) &\overset{8}{=} \mathsf{Inst}_Y(\mathsf{Inst}_{\mathcal{P}\setminus\{Y\}}(\mathcal{E})) \overset{(IH)}{=} (\mathsf{Inst}_Y \circ \bigcirc_{X\in\mathcal{P}\setminus\{Y\}}\mathsf{Inst}_X)(\mathcal{E}) \\
&= (\bigcirc_{X\in\mathcal{P}}\mathsf{Inst}_P)(\mathcal{E}). \qquad \square
\end{aligned}$$

Together with the correctness of Inst_X (Theorem 1), the latter result allows us to prove the main theorem of this section, which states correctness of $\mathsf{Inst}_{\mathcal{P}}$.

Theorem 2. *Let $\mathcal{E}$ be a PBES and $\mathcal{P} \subseteq \mathsf{bnd}(\mathcal{E})$ be a set of predicate variables that is instantiation-fresh for $\mathcal{E}$. Then for all environments η, ε:*

(a) $\forall X \in \mathcal{P}\,.\,\forall v \in D\,.\,([\![\mathsf{Inst}_{\mathcal{P}}(\mathcal{E})]\!]\eta\varepsilon)(X_v) = ([\![\mathcal{E}]\!]\eta\varepsilon)(X([\![v]\!]))$

(b) $\forall Y \in \mathcal{X}\,.\,Y \notin \mathsf{all}(\mathcal{P}) \cup \mathcal{P} \implies ([\![\mathsf{Inst}_{\mathcal{P}}(\mathcal{E})]\!]\eta\varepsilon)(Y) = ([\![\mathcal{E}]\!]\eta\varepsilon)(Y)$.

Proof. The proof goes by induction on the size of $\mathcal{P}$, relying on Lemma 9 and Theorem 1 for proving the correctness of instantiating a single variable. $\square$

$$\mathsf{Inst}_\infty(\epsilon) \triangleq \epsilon$$
$$\mathsf{Inst}_\infty((\sigma X(d{:}D) = \varphi)\ \mathcal{E}) \triangleq \{(\sigma X_v = \mathsf{Sub}_\infty(\varphi[d := v])) \mid v \in D\}\ \mathsf{Inst}_\infty(\mathcal{E})$$

where

$$\begin{aligned}
\mathsf{Sub}_\infty(b) &\triangleq \mathsf{eval}(b)\\
\mathsf{Sub}_\infty(X(d)) &\triangleq \bigvee_{v\in D} (\mathsf{eval}(v = d) \wedge X_v)\\
\mathsf{Sub}_\infty(\varphi_1 \oplus \varphi_2) &\triangleq \mathsf{Sub}_\infty(\varphi_1) \oplus \mathsf{Sub}_\infty(\varphi_2)\\
\mathsf{Sub}_\infty(\mathsf{Q}d{:}D\,.\,\varphi) &\triangleq \bigoplus_{v\in D} \mathsf{Sub}_\infty(\varphi[d := v])
\end{aligned}$$

where $\bigoplus = \bigwedge$ if $\mathsf{Q} = \forall$, and $\bigoplus = \bigvee$ if $\mathsf{Q} = \exists$.

Fig. 2. The instantiation method for countable domains Inst_∞.

The above result allows for a full instantiation of a PBES to a BES. This is a sound strategy when (1) all data sorts that occur in the PBES are finite, (2) the PBES is closed and data-closed, and (3) it is possible to rewrite every data term that occurs in a right-hand side of the PBES to either $\top$ or $\bot$. We assume that the latter can be achieved by a data term evaluator eval, which can be implemented using *e.g.* rewriting technology, see also Section 6; the data term evaluator can be lifted to PBESs in a straightforward manner. The following is then a corollary to Lemma 9.

Corollary 1. *Let $\mathcal{E}$ be a PBES. If $\mathcal{E}$ is closed and data-closed, all data sorts occurring in $\mathcal{E}$ are finite, and a suitable term rewriter* eval *exists, then the equation system* $\mathsf{eval}(\mathsf{Inst}_{\mathsf{bnd}(\mathcal{E})}(\mathcal{E}))$ *is a BES.*

We provide a small example to illustrate the transformation performed by $\mathsf{Inst}_{\mathcal{P}}$.

Example 5. Consider the following PBES $\mathcal{E}$:

$$\begin{aligned}
\nu X(b{:}B) &= \ b \wedge Y(\neg b)\\
\mu Y(b{:}B) &= \neg b \vee X(b).
\end{aligned}$$

Instantiation of X and Y in $\mathcal{E}$ yields the BES $\mathcal{E}'$ below, after minor rewriting:

$$(\nu X_\top = Y_\bot)\ (\nu X_\bot = \bot)\ (\mu Y_\top = X_\top)\ (\mu Y_\bot = \top).$$

The BES $\mathcal{E}'$ can be solved using substitution and approximation, by which we obtain the following correspondence between $\mathcal{E}$ and $\mathcal{E}'$, for any $b \in B$ and environments η and ε:

- $[\![\mathcal{E}]\!]\eta\varepsilon(X)(b) = [\![\mathcal{E}']\!]\eta\varepsilon(X_b) = [\![b]\!]$ and
- $[\![\mathcal{E}]\!]\eta\varepsilon(Y)(b) = [\![\mathcal{E}']\!]\eta\varepsilon(Y_b) = \top$.

This concludes our treatment of instantiation on finite domains. In the next section, we consider instantiation on countably infinite domains.

5. Instantiation on countable domains

In the previous section, we assumed that the domain D of the instantiated variable was finite. Instantiation then resulted in a PBES in which the predicate variables still carried parameters with a (possibly) infinite domain. In this section, we lift the restriction of finiteness and consider PBESs in which each predicate variable is either of type $D \to B$ or of type B, where D is a possibly infinite, yet countable sort. With each predicate variable $X : D \to B$, we associate a countable set of proposition variables $\mathsf{all}(X) \triangleq \{X_d{:}B \mid d \in D\}$.

The instantiation method [1] is listed in Fig. 2; it generates an IBES from a PBES. For every equation $\sigma X(d{:}D) = \varphi$ in the PBES, a block of countably many equations is generated, each of which is of the form $\sigma X_v = \omega_v$ for some $v \in D$ and infinite proposition formula $\omega_v = \mathsf{Sub}_\infty(\varphi[d := v])$. To ensure that every ω_v is indeed a proper infinite proposition formula, we rely on the term evaluator eval to rewrite every data term in $\varphi[d := v]$ to either $\top$ or $\bot$. Hence, $\varphi[d := v]$ must be closed implying that φ may contain no free data variables other than d. This is ensured by allowing only data-closed PBESs for method Inst_∞.

The main correspondence between the predicate variables of a PBES and the proposition variables of the IBES resulting from the instantiation, is given in Theorem 3. Below, we first lift Lemma 3 to countable domains.

[1] We do not use the term "algorithm" as our method does not necessarily terminate.

Lemma 10. *Let φ be a closed predicate formula and η_1, η_2 be environments such that $\forall X \in \mathsf{occ}(\varphi) \,.\, \forall v \in D \,.\, \eta_1(X)([\![v]\!]) = \eta_2(X_v)$. Then for any environment ε:*

$$[\![\varphi]\!]\eta_1\varepsilon = [\![\mathsf{Sub}_\infty(\varphi)]\!]\eta_2.$$

Proof. The proof is similar to the proof of Lemma 3 and is therefore omitted. □

Theorem 3. *Let $\mathcal{E}$ be a data-closed PBES such that every $X \in \mathsf{var}(\mathcal{E})$ is instantiation-fresh for $\mathcal{E}$, and let η be an environment satisfying:*

$$\forall Y \in \mathsf{occ}(\mathcal{E})\backslash\mathsf{bnd}(\mathcal{E}) \,.\, \forall w \in D \,.\, \eta(Y_w) = \eta(Y)([\![w]\!]). \tag{2}$$

Then, for any environment ε:

$$\forall X \in \mathsf{bnd}(\mathcal{E}) \,.\, \forall v \in D \,.\, ([\![\mathsf{Inst}_\infty(\mathcal{E})]\!]\eta)(X_v) = ([\![\mathcal{E}]\!]\eta\varepsilon)(X)([\![v]\!]).$$

Proof. Let ε be an environment. The proof goes by induction on the length of $\mathcal{E}$. If $\mathcal{E} = \epsilon$, the statement holds vacuously. For the inductive case we assume, for all PBESs $\mathcal{E}'$ of length m for which all variables are instantiation-fresh and environments η', ε' satisfying (2):

$$\forall X \in \mathsf{bnd}(\mathcal{E}') \,.\, \forall v \in D \,.\, ([\![\mathsf{Inst}_\infty(\mathcal{E}')]\!]\eta')(X_v) = ([\![\mathcal{E}']\!]\eta'\varepsilon')(X)([\![v]\!]). \tag{IH}$$

Suppose $\mathcal{E}$ is of length $m+1$, so $\mathcal{E} = (\sigma Y(d{:}D) = \varphi)\ \mathcal{E}'$ for some PBES $\mathcal{E}'$ of length m. We define the following shorthands:

$$\begin{aligned}
\sigma\mathcal{B} &\triangleq \{\sigma Y_w = \mathsf{Sub}_\infty(\varphi[d := w]) \mid w \in D\} \\
f &\triangleq \sigma g \in \mathbb{B}^D \,.\, \lambda w \in D \,.\, [\![\mathsf{Sub}_\infty(\varphi[d := w])]\!]([\![\mathsf{Inst}_\infty(\mathcal{E}')]\!]\eta[Y_D \mapsto g]) \\
h &\triangleq \sigma k \in \mathbb{B}^{\mathbb{D}} \,.\, \lambda w \in \mathbb{D} \,.\, [\![\varphi]\!]([\![\mathcal{E}']\!]\eta[Y \mapsto k]\varepsilon)\varepsilon[d \mapsto w].
\end{aligned}$$

Let $X \in \mathsf{bnd}(\mathcal{E})$ and $v \in D$. Then:

$$\begin{aligned}
&[\![\mathsf{Inst}_\infty((\sigma Y(d{:}D) = \varphi)\ \mathcal{E}')]\!]\eta(X_v) \\
&\quad = [\![\sigma\mathcal{B}\ \mathsf{Inst}_\infty(\mathcal{E}')]\!]\eta(X_v) \\
&\quad = [\![\mathsf{Inst}_\infty(\mathcal{E}')]\!]\eta[Y_D \mapsto f](X_v) \\
&\quad = [\![\mathsf{Inst}_\infty(\mathcal{E}')]\!]\eta[Y_D \mapsto f][Y \mapsto h](X_v) \\
&\quad \overset{*}{=} ([\![\mathcal{E}']\!]\eta[Y_D \mapsto f][Y \mapsto h]\varepsilon)(X)([\![v]\!]) \\
&\quad = ([\![\mathcal{E}']\!]\eta[Y \mapsto h]\varepsilon)(X)([\![v]\!]) \\
&\quad = ([\![(\sigma Y(d{:}D) = \varphi)\ \mathcal{E}']\!]\eta\varepsilon)(X)([\![v]\!]).
\end{aligned}$$

At $*$ we used (IH) for which we need to prove that:

$$\forall X \in \mathsf{var}(\mathcal{E}') \,.\, \mathsf{all}(X) \cap \mathsf{var}(\mathcal{E}') = \emptyset \tag{3}$$

$$\forall Z \in \mathsf{occ}(\mathcal{E}')\backslash\mathsf{bnd}(\mathcal{E}') \,.\, \forall x \in D \,. \quad \eta[Y_D \mapsto f][Y \mapsto h](Z_x) = \eta[Y_D \mapsto f][Y \mapsto h](Z)([\![x]\!]). \tag{4}$$

Property (3) follows from the facts that all variables in $\mathcal{E}$ are instantiation-fresh and $\mathsf{var}(\mathcal{E}') \subseteq \mathsf{var}(\mathcal{E})$. Regarding (4), let $Z \in \mathsf{occ}(\mathcal{E}')\backslash\mathsf{bnd}(\mathcal{E}')$ and $x \in D$. If $Z \neq Y$ then observe that $Z \in \mathsf{occ}(\mathcal{E})\backslash\mathsf{bnd}(\mathcal{E})$. Then because $\mathcal{E}$ and η satisfy (2): $\eta[Y_D \mapsto f][Y \mapsto h](Z_x) = \eta(Z_x) = \eta(Z)([\![x]\!]) = \eta[Y_D \mapsto f][Y \mapsto h](Z)([\![x]\!])$.

If $Z = Y$ then $\eta[Y_D \mapsto f][Y \mapsto h](Y_x) = f(x)$ and $\eta[Y_D \mapsto f][Y \mapsto h](Y)([\![x]\!]) = h([\![x]\!])$, so we need to prove that $f(x) = h([\![x]\!])$. Let $g : D \to \mathbb{B}$ and for that g define $k : \mathbb{D} \to \mathbb{B}$ as follows: $k([\![d]\!]) = g(d)$ for all $d \in D$. Then $f(x) = h([\![x]\!])$ follows if:

$$\begin{aligned}
&(\lambda w \in D \,.\, [\![\mathsf{Sub}_\infty(\varphi[d := w])]\!]([\![\mathsf{Inst}_\infty(\mathcal{E}')]\!]\eta[Y_D \mapsto g]))(x) \\
&\quad = (\lambda w \in \mathbb{D} \,.\, [\![\varphi]\!]([\![\mathcal{E}']\!]\eta[Y \mapsto k]\varepsilon)\varepsilon[d \mapsto w])([\![x]\!]).
\end{aligned}$$

We derive, starting at the right-hand side:

$$\begin{aligned}
&(\lambda w \in \mathbb{D} \,.\, [\![\varphi]\!]([\![\mathcal{E}']\!]\eta[Y \mapsto k]\varepsilon)\varepsilon[d \mapsto w])([\![x]\!]) \\
&\quad = [\![\varphi]\!]([\![\mathcal{E}']\!]\eta[Y \mapsto k]\varepsilon)\varepsilon[d \mapsto [\![x]\!]] \\
&\quad = [\![\varphi[d := x]]\!]([\![\mathcal{E}']\!]\eta[Y \mapsto k]\varepsilon)\varepsilon \\
&\quad = [\![\varphi[d := x]]\!]([\![\mathcal{E}']\!]\eta[Y_D \mapsto g][Y \mapsto k]\varepsilon)\varepsilon
\end{aligned}$$

$$\stackrel{*}{=} [\![\mathsf{Sub}_\infty(\varphi[d := x])]\!]([\![\mathsf{Inst}_\infty(\mathcal{E}')]\!]\eta[Y_D \mapsto g][Y \mapsto k])$$
$$= [\![\mathsf{Sub}_\infty(\varphi[d := x])]\!]([\![\mathsf{Inst}_\infty(\mathcal{E}')]\!]\eta[Y_D \mapsto g])$$
$$= (\lambda w \in D \,.\, [\![\mathsf{Sub}_\infty(\varphi[d := w])]\!]([\![\mathsf{Inst}_\infty(\mathcal{E}')]\!]\eta[Y_D \mapsto g]))(x).$$

For convenience we define $\theta \stackrel{\Delta}{=} \eta[Y_D \mapsto g][Y \mapsto k]$. At $*$ we used Lemma 10 which is allowed because $\varphi[d := x]$ is closed (by data-closedness of $\mathcal{E}$) and we have:

$$\forall W \in \mathsf{occ}(\varphi) \,.\, \forall w \in D \,.\, ([\![\mathcal{E}']\!]\theta\varepsilon)(W)([\![w]\!]) = ([\![\mathsf{Inst}_\infty(\mathcal{E}')]\!]\theta)(W_w)$$

which we prove now. Let $W \in \mathsf{occ}(\varphi)$ and $w \in D$. There are three cases:

1. $W = Y$. Then $([\![\mathcal{E}']\!]\theta\varepsilon)(Y)([\![w]\!]) = k([\![w]\!]) = g(w) = ([\![\mathsf{Inst}_\infty(\mathcal{E}')]\!]\theta)(Y_w)$.
2. $W \neq Y$ and $W \notin \mathsf{bnd}(\mathcal{E})$. Then:

   $$([\![\mathcal{E}']\!]\theta\varepsilon)(W)([\![w]\!]) = \eta(W)([\![w]\!]) \stackrel{\dagger}{=} \eta(W_w) \stackrel{\ddagger}{=} ([\![\mathsf{Inst}_\infty(\mathcal{E}')]\!]\theta)(W_w).$$

   At † we used the fact that $\mathcal{E}$ and η satisfy (2) in combination with $W \in \mathsf{occ}(\mathcal{E})$ and $W \notin \mathsf{bnd}(\mathcal{E})$. At ‡ we used Lemma 1 combined with $W_w \notin \mathsf{bnd}(\mathsf{Inst}_\infty(\mathcal{E}'))$.
3. $W \neq Y$ and $W \in \mathsf{bnd}(\mathcal{E})$. Observe that $W \in \mathsf{bnd}(\mathcal{E}')$. Then the equivalence follows from (IH) if we prove that all variables in $\mathcal{E}'$ are instantiation-fresh and θ satisfies (2). The former follows from the fact that all variables in $\mathcal{E}$ are instantiation-fresh and $\mathsf{var}(\mathcal{E}') \subseteq \mathsf{var}(\mathcal{E})$. The latter follows using similar reasonings as in cases 1 and 2. □

From Theorem 3 we obtain the following result.

Corollary 2. *Let $\mathcal{E}$ be a PBES. If $\mathcal{E}$ is closed and data-closed, and all data sorts occurring in $\mathcal{E}$ are countable and a suitable term rewriter* eval *exists, then* $\mathsf{Inst}_\infty(\mathcal{E})$ *is an IBES.*

We remark that for typical verification problems, such as (local) model checking and equivalence checking, a partial solution to the PBES is often satisfactory. Using Proposition 1, it is straightforward to turn the instantiation scheme for PBESs involving countable data sorts into a procedure for computing a BES; see Section 6.

Example 6. Consider the following PBES $\mathcal{E}$:

$$\nu X(n{:}N) = n \neq 1 \wedge X(n+1).$$

Applying the transformation Inst_∞, we obtain the following IBES:

$$\{(\nu X_0 = X_1)\ (\nu X_1 = \bot)\ (\nu X_n = X_{n+1}) \mid n \geq 2\}.$$

While solving, *e.g.* $X(5)$ by means of a transformation to IBES would require an infinite computation, solving $X(0)$ or $X(1)$ would terminate using a local resolution: $X(0)$ depends on $X(1)$ which is immediately $\bot$.

Note that the transformation from an IBES to a PBES is elementary, provided one has a sufficiently rich data language: the sorts that are used for the blocks in the IBES can be introduced as the data sorts of the PBES, and the infinite disjunction and conjunction that occur in the infinite proposition formulae can be converted to equality tests and universal and existential quantifications, respectively.

6. Implementation

The predominant factor in the undecidability of our verification problems lies in the use of first-order constructs in our μ-calculus and process language. In this section, we address the problems that surface when implementing the instantiation methods of the previous sections.

6.1. On-the-fly exploration

As mentioned in the previous section, for local model checking, we can resort to an on-the-fly exploration of an IBES $\mathsf{Inst}_\infty(\mathcal{E})$ to try and answer whether the solution to some $X(e)$ is either $\top$ or $\bot$. For this, we require an auxiliary function pvi that, given a predicate formula, yields the set of *predicate variable instantiations* occurring in the formula. Formally, we have, for any formulae φ_1, φ_2:

$$\mathsf{pvi}(b) \stackrel{\Delta}{=} \emptyset \qquad \mathsf{pvi}(X(e)) \stackrel{\Delta}{=} \{X(e)\}$$
$$\mathsf{pvi}(\varphi_1 \oplus \varphi_2) \stackrel{\Delta}{=} \mathsf{pvi}(\varphi_1) \cup \mathsf{pvi}(\varphi_2) \qquad \mathsf{pvi}(\mathsf{Q}d{:}D.\varphi_1) \stackrel{\Delta}{=} \mathsf{pvi}(\varphi_1).$$

Observe that for formulae φ containing proposition variables only, $\mathsf{pvi}(\varphi) = \mathsf{occ}(\varphi)$. A pseudo-code implementation of a procedure that performs an on-the-fly instantiation of a given PBES is PBES2BES (Procedure 1).

Procedure 1 On-the-fly instantiation (Procedure PBES2BES).

Require:
$\mathcal{E} \equiv (\sigma_1 X_1(d_1{:}D_1) = \varphi_1) \cdots (\sigma_n X_n(d_n{:}D_n) = \varphi_n)$
$\mathcal{E}$ is closed
$1 \leq i \leq n$

```
1:  procedure PBES2BES(E, X_i(e), R, ρ)
2:      for each j ∈ {1, ..., n} do E_j := ε;
3:      found, done := {R(X_i(e))}, ∅;
4:      while found ≠ done do
5:          choose X_k(e_k) ∈ found\done;
6:          X̃, φ̃ := ρ(X_k(e_k)), R(φ_k[d_k := e_k]);
7:          E_k := E_k (σ_k X̃ = ρ(φ̃));
8:          found, done := found ∪ pvi(φ̃), done ∪ {X_k(e_k)};
9:      end while
10:     return E_1 ··· E_n
11: end procedure
```

Apart from the PBES $\mathcal{E}$ and some closed predicate variable instantiation $X_i(e)$, procedure PBES2BES requires two additional arguments, *viz.*, a *rewriter* R and a *renaming function* ρ. The latter is required to be an injective function, mapping closed expressions $X(e)$ to a proposition variable; ρ easily extends to quantifier-free predicate formulae by defining $\rho(b) \stackrel{\Delta}{=} b$ and $\rho(\varphi_1 \oplus \varphi_2) \stackrel{\Delta}{=} \rho(\varphi_1) \oplus \rho(\varphi_2)$. The rewriter R is assumed to rewrite every closed data expression to a basic element of the same sort; in this sense, it generalises the term evaluator eval of the previous section. On top of this, we assume R is capable of rewriting (possibly open) predicate formulae. In addition to these basic requirements, we assume that R can be used to eliminate quantifiers; we will come back to this assumption in Section 6.2. Formally, we require R to be sound, *i.e.* $R(\bot) = \bot$ and for data expressions t, u, if $R(t) = R(u)$, then $[\![t]\!]\varepsilon = [\![u]\!]\varepsilon$ for all data environments ε. Under these assumptions, we have the following theorem.

Theorem 4. *Let $\mathcal{E}$ be a closed PBES. Let $X{:}D \to \mathbb{B} \in \mathsf{bnd}(\mathcal{E})$. Let e be a closed data term of sort D. Assume ρ is a suitable renaming function and R a suitable rewriter. Upon termination, procedure* PBES2BES *returns a BES $\mathcal{E}_1 \cdots \mathcal{E}_n$ satisfying:*

$$[\![\mathcal{E}]\!]\eta\varepsilon(X([\![e]\!])) = [\![\mathcal{E}_1 \cdots \mathcal{E}_n]\!]\eta(\rho(X(e))).$$

Proof. We sketch the proof. Observe that procedure PBES2BES satisfies the following three invariants:

1. $done \subseteq found$;
2. $\mathsf{bnd}(\mathcal{E}_1 \cdots \mathcal{E}_n) = \rho(done)$;
3. $\mathsf{occ}(\mathcal{E}_1 \cdots \mathcal{E}_n) \subseteq \rho(found)$.

Upon termination, $\rho(done) = \rho(found)$, which, combined with the above invariants, proves that $\mathsf{occ}(\mathcal{E}_1 \cdots \mathcal{E}_n) \subseteq \mathsf{bnd}(\mathcal{E}_1 \cdots \mathcal{E}_n)$, *i.e.* $\mathcal{E}_1 \cdots \mathcal{E}_n$ is closed. Each equation in $\mathcal{E}_1 \cdots \mathcal{E}_n$ can, moreover, be related to an equation in $\mathsf{Inst}_\infty(\mathcal{E})$. Combining Theorem 3 with (a repeated application of) Proposition 1, we find that the required equivalence holds. □

Observe that the inner loop in procedure PBES2BES can be implemented using, *e.g.* a depth-first or a breadth-first exploration.

6.2. Eliminating quantifiers

One of the main challenges in rewriting in our setting is dealing with quantified variables ranging over countable data sorts. While rewriting is no longer guaranteed to terminate when countable data sorts are involved, a pragmatic approach to rewriting may often be successful. Quantifiers, however, are a major obstacle, as illustrated by the following equation, taken from a more complex equation system:

$$\nu X(n{:}N) = \forall m{:}N.\ m \leq n + 10 \implies Y(m)$$

Starting the on-the-fly instantiation at $X(0)$ requires the computation of the equations for $Y(0)$ up-to and including $Y(10)$. Finding these dependencies, however, requires a strategy that takes advantage of the structure of the infinite data sort N and the fact that for $m > 10$, the right-hand side expression of the equation always evaluates to $\top$. Below, we discuss a procedure, called EliminateQuantifier, that tries to eliminate a single quantifier from a predicate formula. First, we elaborate on the theory behind this procedure.

Procedure 2 Elimination of a single quantification (Procedure EliminateQuantifier).

```
1: procedure EliminateQuantifier (Qd:D. φ, R)
2:   if d ∉ dvar(φ) then return R(φ)
3:   else
4:     V, W := ∅, {d};
5:     repeat
6:       choose e ∈ W;
7:       W := W\{e};
8:       for t ∈ enum_φ(e) do
9:         if dvar(R(φ[d := t])) ⊆ dvar(φ)\{d} then
10:          V := V ∪ {t}
11:        else
12:          W := W ∪ {t}
13:        end if
14:      end for
15:    until W = ∅
16:  end if
17:  return ⨁{R(φ[d := t]) | t ∈ V}
18: end procedure
```

Let φ be an arbitrary predicate formula. We denote the set of *all* data variables occurring in φ, bound and unbound, by $\mathsf{dvar}(\varphi)$. For a set of formulae Φ, $\mathsf{dvar}(\Phi)$ is defined as the union of $\mathsf{dvar}(\varphi)$, for all $\varphi \in \Phi$. Let D be a countable data sort, and let T be a finite set of (possibly open) terms of sort D. We say T *spans* D iff $\bigcup_\varepsilon \{[\![t]\!]\varepsilon \mid t \in T\} = \mathbb{D}$. In words, by choosing appropriate values for the free variables in T, we are able to construct every element of D. We have the following proposition:

Proposition 2. *Let $\mathsf{Q}d{:}D.\varphi$ be an arbitrary formula. For each finite set of terms T spanning D, with $\mathsf{dvar}(T) \cap \mathsf{dvar}(\varphi) = \emptyset$, we have the following correspondence:*

$$\forall \eta, \varepsilon \,.\, [\![\mathsf{Q}d{:}D.\ \varphi]\!]\eta\varepsilon = \bigoplus\{[\![\varphi]\!]\eta\varepsilon[d \mapsto [\![t]\!]\varepsilon'] \mid t \in T \text{ and arbitrary } \varepsilon'\}$$

where $\bigoplus = \bigwedge$ if $\mathsf{Q} = \forall$ and $\bigoplus = \bigvee$ otherwise.

As a consequence of the above correspondence, it suffices to search for a set of terms T spanning sort D, with the property that for each term $t \in T$, rewriter R reduces the expression $\varphi[d := t]$ to an expression such that $\mathsf{dvar}(R(\varphi[d := t])) \subseteq \mathsf{dvar}(\varphi)\backslash\{d\}$. In words, by substituting the (possibly open) terms $t \in T$ for variable d in φ, variable d and all fresh variables possibly introduced by the substitution, can be removed by rewriting. In case such a T is found, we know that $[\![\mathsf{Q}d{:}D.\ \varphi]\!]\eta\varepsilon = \bigoplus\{[\![R(\varphi[d := t])]\!]\eta\varepsilon \mid t \in T\}$, which follows from Proposition 2 and the soundness of rewriter R.

Procedure EliminateQuantifier maintains two sets of expressions V and W, of sort D. Together these span sort D. At each iteration, an expression in set W is selected for refinement. The refinement is based on a decomposition of the free data variables occurring in the expression, which is given by the function enum_φ. This function, when given a term e, yields a set E of new expressions (with $\mathsf{dvar}(E) \cap \mathsf{dvar}(\varphi) = \emptyset$) that collectively represent all possible semantic data values of expression e. Formally, we assume enum_φ to satisfy the following properties for all data expressions e of sort D:

1. $0 < |\mathsf{enum}_\varphi(e)| < \infty$;
2. $\bigcup_\varepsilon \{[\![e]\!]\varepsilon\} = \bigcup_\varepsilon \{[\![e']\!]\varepsilon \mid e' \in \mathsf{enum}_\varphi(e)\}$.

In our context of abstract data types, we use an implementation of enum_φ that relies on the constructors of a sort to produce a finite set of new expressions, since the set of constructor functions of a sort is finite and adequate for representing all possible basic elements of that sort.

Example 7. Henceforth, assume that the sort N is defined by the constructors $0{:}N$ and $s{:}N \to N$. A first refinement of a data expression consisting of only a single variable $m{:}N$ using $\mathsf{enum}_\varphi(m)$ would yield $\{0, s(n_0)\}$, and a successive refinement of the data expression $s(n_0)$ would yield $\{s(0), s(s(n_1))\}$, whereas 0 would be refined by $\{0\}$ only.

Lastly, remark that the loop in EliminateQuantifier satisfies the following invariants:

1. $V \cup W$ spans D;
2. $\mathsf{dvar}(V \cup W) \cap \mathsf{dvar}(\varphi) = \emptyset$;
3. For each $t \in W$, $\mathsf{dvar}(t) \neq \emptyset$;
4. For each $t \in V$, $\mathsf{dvar}(R(\varphi[d := t])) \subseteq \mathsf{dvar}(\varphi)\backslash\{d\}$.

Theorem 5. *Let* $\mathsf{Q}d{:}D.\ \varphi$ *be an arbitrary predicate formula. Upon termination, procedure* EliminateQuantifier *returns a predicate formula* ψ*, satisfying:*

1. $\mathsf{dvar}(\psi) \subseteq \mathsf{dvar}(\varphi)\backslash\{d\}$*, and*
2. $[\![\mathsf{Q}d{:}D.\ \varphi]\!]\eta\varepsilon = [\![\psi]\!]\eta\varepsilon$.

Proof. Follows immediately from the loop-invariants and Proposition 2. □

It is important to note that procedure EliminateQuantifier can already be terminated if a refinement leads to the complement of the unit of the quantifier. For instance, if for some $t \in \mathsf{enum}_\varphi(e)$, $R(\varphi[d := t]) = \top$, then $\exists d{:}D.\varphi$ is equivalent to $\top$, so no further refinements are needed. Likewise, if $R(\varphi[d := t]) = \bot$, then $\forall d{:}D.\varphi$ is equivalent to $\bot$.

In theory, procedure EliminateQuantifier can be applied recursively if rewriter R is extended with the procedure. However, doing so naively may not lead to termination in cases where termination could be achieved practically. The example below illustrates such a situation.

Example 8. Consider the formula $\forall b{:}B.\ \forall i{:}N.(i \geq j \wedge b)$, where j is some freely occurring variable. Intuitively, the formula should rewrite to $\bot$. Assume the constructors $\top{:}B$ and $\bot{:}B$ for the Booleans, and $0{:}N$ and $s{:}N \to N$ for the natural numbers.

Running procedure EliminateQuantifier on the above formula recursively leads to a refinement of expression i that will not terminate. This can be seen as follows. Elimination of variable b will recursively call EliminateQuantifier on expressions $\forall i{:}N.\ (i \geq j \wedge \bot)$, which will lead to $\bot$, and expression $\forall i{:}N.\ (i \geq j \wedge \top)$. The latter call will not terminate, since, independent of the choice for enum, there is always some $e \in \mathsf{enum}(i)$ such that $\mathsf{dvar}(R(e \geq j \wedge \top)) \supset \{j\}$. Note that termination is impossible even if one allows for early termination of EliminateQuantifier, since none of the terms $R(e \geq j \wedge \top)$ can reduce to $\bot$, unless the value for j is known.

The problem that manifests itself in the example above is that a straightforward recursive application of EliminateQuantifier can give rise to a deeply nested non-terminating sequence of refinements. Assuming the presence of suitable pairing and projection functions π_i, the above situation can be avoided by utilising the following correspondences, allowing procedure EliminateQuantifier to be run on multiple variables.

$$\begin{aligned}\forall \eta, \varepsilon\ .\ &= [\![\mathsf{Q}d_1{:}D_1.\ \cdots \mathsf{Q}d_n{:}D_n.\ \varphi]\!]\eta\varepsilon \\ &= [\![\mathsf{Q}(d_1, \ldots, d_n){:}D_1 \times \cdots \times D_n.\ \varphi]\!]\eta\varepsilon \\ &= [\![\mathsf{Q}\vec{d}{:}D_1 \times \cdots \times D_n.\ \varphi[d_1, \ldots, d_n := \pi_1(\vec{d}), \ldots, \pi_n(\vec{d})]]\!]\eta\varepsilon\end{aligned}$$

Below, we illustrate how this can be applied for the formula of Example 8.

Example 9. Consider again formula $\forall b{:}B.\ \forall i{:}N.(i \geq j \wedge b)$. Observe that this formula is logically equivalent to $\forall (b, i){:}B \times N.(i \geq j \wedge b)$, which again is logically equivalent to $\forall \vec{c}{:}B \times N.\ (\pi_2(\vec{c}) \geq j \wedge \pi_1(\vec{c}))$. Applying procedure EliminateQuantifier on the latter formula requires us to refine expression $\vec{c}$ in the first step. For clarity, we use variables b and i instead, rather than writing the "abstract" $\vec{c}$, and the projections $\pi_1(\vec{c})$ and $\pi_2(\vec{c})$. Assuming that $\mathsf{enum}((b, i)) = \{(\bot, 0), (\top, 0), (\bot, s(i')), (\top, s(i'))\}$, it is easy to see that using the substitutions $(\bot, 0)$ and $(\bot, s(i'))$ for (b, i), the formula can be reduced to $\bot$. This will allow the procedure to terminate at this point. Note that if termination is not enforced here, the refinements $(\bot, 0)$, $(\bot, s(i'))$ and $(\top, 0)$ end up in set V and the refinement $(\top, s(i'))$ winds up in set W. The procedure will continue to explore all refinements of $(\top, s^n(i'))$ in much the same fashion, without ever reaching the conclusion that $W = \emptyset$.

Remark 1. Procedure EliminateQuantifier employs a *breadth-first* exploration of refinements of terms (lines 8–14). Alternatively, a procedure employing a *depth-first* exploration can be defined, in which EliminateQuantifier is recursively called from within lines 8–14; this way, set W needs not be maintained explicitly. While the latter can be quicker at times, especially in combination with some heuristics to select the sequences of refinements that should be traversed first, it is also less often guaranteed to terminate than the breadth-first exploration. The formula of Example 8 is such an example: the depth-first exploration may pursue the infinite sequence of refinements of variable i, under the presumption that b is $\top$.

Procedures EliminateQuantifier and PBES2BES have been implemented in C++ as part of the mCRL2 toolset. It builds upon the rewriting engines included in this toolset. For details on the rewriting technology that is implemented in mCRL2 we refer to [34].

7. Examples

In this section, we further demonstrate the instantiation techniques of the previous sections by applying them to several examples. Two prime – but lengthy – example applications of the manipulation are already contained in the full version

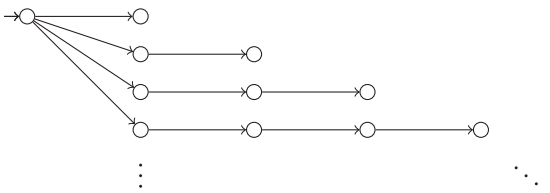


Fig. 3. An infinite transition system in which every path from the initial state is of finite length.

of [5]. First, we demonstrate that the partial instantiation of a PBES is a useful manipulation in itself. Next, we illustrate the feasibility of instantiating a given PBES to an (I)BES.

7.1. Model-checking infinite-state systems

Two smaller examples, derived from model-checking problems on infinite state systems, are given below. These problems first appeared in [35] and were revisited in [8] to demonstrate the efficacy of IBESs.

7.1.1. Checking for finite paths

Consider the following LPE:

$$P(b{:}B, n{:}N) = \textstyle\sum_{i:N} b \to a \cdot P(\neg b, i) + \\ \neg b \wedge n > 0 \to a \cdot P(b, n-1)$$

with initial state $P(\top, 0)$. Its transition system is infinitely large and (partially) depicted in Fig. 3, where the a-labels have been omitted. The property that Bradfield [35] and Mader [8] verify is that every path that starts in the initial state has finite length only. This property is expressed by the following μ-calculus formula: $\mu X \,.\, [\top]X$. Note that the number of paths in the system is infinite.

The following PBES, consisting of a single equation, encodes the above model-checking problem, where satisfaction of the property by the initial state corresponds with $X(\top, 0)$:

$$\mu X(b{:}B, n{:}N) = (\forall i{:}N \,.\, \neg b \vee X(\neg b, i)) \wedge (b \vee n = 0 \vee X(b, n-1)).$$

A solution technique based on a straightforward symbolic approximation as described in *e.g.* [1] does not terminate. Patterns [4] for solving PBESs, which allow one to "look up" a solution for equations of a particular shape, are also not applicable. Instantiation of X on the parameter b leads to the following PBES:

$$\mu X_\bot(n{:}N) = (n = 0) \vee X_\bot(n-1)$$
$$\mu X_\top(n{:}N) = \forall i{:}N \,.\, X_\bot(i).$$

Now, the equation for $X_\bot$ can easily be solved by means of a pattern, which leads to the following equivalent equation system:

$$\mu X_\bot(n{:}N) = \exists i{:}N \,.\, n = i$$
$$\mu X_\top(n{:}N) = \forall i{:}N \,.\, X_\bot(i).$$

Using standard logic, the above equation system can immediately be rewritten (even automatically [4]) to the following:

$$\mu X_\bot(n{:}N) = \top$$
$$\mu X_\top(n{:}N) = \top.$$

Hence the property holds for all reachable states, and the initial state in particular. The proof in [8] requires a manual construction of a set-based representation of an IBES, and requires showing the well-foundedness of mappings of this representation. The tableaux-based methods of Bradfield [35] require the investigation of *extended paths*. Our proof strategy, using partial instantiation, is easier to understand and requires less effort.

7.1.2. Checking for a finite number of actions

Consider the following LPE:

$$P(b{:}B, n{:}N) = b \to c \cdot P(\neg b, n) + b \to a \cdot P(b, n+1) + \neg b \wedge n > 0 \to a \cdot P(b, n-1)$$

with initial state $P(\top, 0)$. This system originated from a Petri net given by Bradfield [35], and reappeared in [8] as a transition system. The LTS of P is depicted in Fig. 4. The property to be verified is that every behaviour in the transition system exhibits only a finite number of c actions: $\mu X \,.\, \nu Y \,.\, ([c]X \wedge [\neg c]Y)$. Note that this formula has alternation depth two.

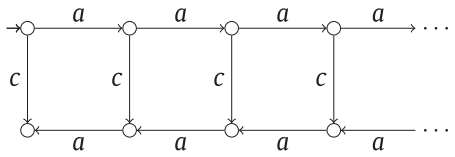


Fig. 4. An infinite transition system that can perform only a finite number of c-steps.

The following PBES, consisting of two equations, encodes the above model-checking problem, where satisfaction of the formula by the initial state corresponds with $X(\top, 0)$:

$$\mu X(b{:}B, n{:}N) = Y(b, n)$$
$$\nu Y(b{:}B, n{:}N) = (\neg b \vee X(\neg b, n)) \wedge (\neg b \vee Y(b, n{+}1)) \wedge (b \vee n{=}0 \vee Y(b, n{-}1)).$$

The above equation system can be solved using a complex pattern which would require the introduction of an auxiliary selector function. Instantiation of Y on parameter b leads to the following PBES:

$$\mu X(b{:}B, n{:}N) = (b \wedge Y_\top(n)) \vee (\neg b \wedge Y_\bot(n))$$
$$\nu Y_\top(n{:}N) = X(\bot, n) \wedge Y_\top(n+1)$$
$$\nu Y_\bot(n{:}N) = (n = 0) \vee Y_\bot(n-1).$$

The equation for $Y_\top$ can now be solved easily by means of a pattern [4]. The equation for $Y_\bot$ is solved instantly using symbolic approximation. This leads to the following equivalent equation system:

$$\mu X(b{:}B, n{:}N) = (b \wedge Y_\top(n)) \vee (\neg b \wedge Y_\bot(n))$$
$$\nu Y_\top(n{:}N) = \forall j{:}N\,.\,X(\bot, n+j)$$
$$\nu Y_\bot(n{:}N) = \top.$$

The solutions to $Y_\bot$ and $Y_\top$ can then be substituted in the equation for X, yielding:

$$\mu X(b{:}B, n{:}N) = (b \wedge \forall j{:}N\,.\,X(\bot, n+j)) \vee \neg b.$$

A symbolic approximation yields the solution $\lambda b{:}B\,.\,\lambda n{:}N\,.\,\top$ for X as the third approximant. Hence the property holds for all reachable states, and the initial state in particular. Again, our proof using partial instantiation is straightforward and enables the use of simple pattern matching, while the earlier proofs by Mader and Bradfield require more effort.

7.2. Automatic verification

In the previous section, we used instantiation to manually solve PBESs that encoded model-checking problems on infinite-state systems. There, the use of instantiation simplified the derivation of the solution considerably, and allowed other techniques to be applied.

To assess the feasibility of automated PBES instantiation in practice, a prototype tool has been implemented that instantiates a given PBES.[2] Upon termination it has generated a BES that holds the answer to whether a particular equation in the original PBES is true for some data value. We apply the tool to verification problems on various models. Most of these models employ the data sort of natural numbers. A full instantiation of the PBES would therefore yield an IBES, but PBES2BES produces a finite BES in all cases.

All experiments were run on a 64-bits architecture computer having an Intel Core2 Quad 2.40 GHz CPU and 4 GB of RAM. It runs Fedora Core 8 Linux, kernel 2.6.26. We use revision 5839 of the mCRL2 toolset.

7.2.1. Checking for deadlocks

We used our tool to check for absence of deadlock on several publicly available benchmarks, consisting of industrial protocols and systems, and games. The deadlock property, defined by $\nu X.[\top]X \wedge \langle\top\rangle\top$, yields an alternation-free PBES. Of course, more involved properties – like fairness and liveness properties – can also be encoded, which may yield PBESs of higher alternation depths, see below. As absence of deadlock requires all reachable states to be computed by the instantiation tool, it allows for a fair comparison with explicit state-space generation. For this, we use the mCRL2 tool lps2lts.

Table 1 contains the results of our experiments for each of the models. It lists the LTS sizes and the times needed to explore these LTSs by lps2lts. The right-most column contains the times needed for instantiating the PBES to a BES *and* solving the

[2] The tool is called pbes2bool and is part of the mCRL2 toolset, see http://www.mcrl2.org.

Table 1
Experimental results for checking for deadlocks in several models using both LTS exploration and PBES instantiation + BES solving.

	LTS size		Time (s)	
Model	States	Transitions	LTS	PBES
BRP	10,548	12,168	5	4
Car lift	4312	9918	7	5
Chatbox	65,536	2,162,688	19	7
IEEE-1394	188,569	340,607	145	134
Clobber	600,161	2,221,553	107	99
Domineering	455,317	2,062,696	56	50
Othello	55,093	88,258	86	104

Table 2
Experimental results for verifying properties I through IV on a Sliding Window Protocol with window size 2 and 4, respectively, 5 different messages. Solving times are the cumulative times required for instantiating the PBES, minimising the BES and solving the resulting BES. The sizes of the LTSs are as follows: for $|D| = 4$ the LTS contains 104,352 states and 588,032 transitions; for $|D| = 5$ the LTS contains 309,300 states and 1,305,300 transitions.

	$\|D\| = 4$		$\|D\| = 5$	
Property	Equations	Solve (s)	Equations	Solve (s)
I	280,705	12	618,601	24
II	163,105	11	349,541	17
III	652,421	26	1,747,706	70
IV	2,245,634	60	6,186,002	162

BES using a combination of approximation and Gauß-elimination. Note that solving only takes place after instantiation has finished. It might be possible to combine solving and instantiation in a clever way, but this was not done for our experiments.

For each model, the number of BES equations after instantiation is equal to the number of states in the corresponding LTS, as expected. The performance of the BES approach is in general comparable to that of the LTS approach; differences are attributed to minor differences in rewriting strategies.

7.2.2. Checking complex requirements

To illustrate the effectiveness of our approach in dealing with more complex requirements, taking advantage of the richness of the first order modal μ-calculus, we consider a verification of a *sliding window protocol* with window size 2 and a set of messages D consisting of 4, respectively, 5 different messages. The properties that we verified are as follows:

I. No livelock, *i.e.* there is no infinite τ-path:
$\nu X.[\top]X \wedge \mu Y.[\tau]Y$.

II. We can infinitely often send a particular message d_0:
$\nu X.\mu Y.\langle s(d_0)\rangle X \vee \langle \neg s(d_0)\rangle Y$.

III. We can send all messages infinitely often:
$\forall d{:}D.\ \nu X.\mu Y.\langle s(d)\rangle X \vee \langle \neg s(d)\rangle Y$.

IV. For each message d, if sending the message is infinitely often enabled, then it is infinitely often sent:
$\forall d{:}D.\ \nu X.[\top]X \wedge \nu Y.\mu Z.\nu W.([s(d)]Y \wedge ([s(d)]\bot \vee [\neg s(d)]Z) \wedge [\neg s(d)]W)$.

The last property is taken from [36], and the PBES encoding this property has an alternation depth of 3. The verification of the above properties leads to the results outlined in Table 2. Note that for solving the resulting BESs, we combined the minimisation techniques described in [13,14] and a (linear time) translation to *Parity Games*. Several competitive algorithms exist for solving the latter; in this case, we have used a freely available implementation[3] of the *bigstep* algorithm due to Schewe [10].

7.2.3. Checking branching bisimilarity

The experiments above illustrate the efficacy of the full instantiation of alternation-free PBESs to (I)BESs. We now consider PBESs in which the branching bisimilarity problem on LPEs is encoded using the translation presented in [5]. The models on which we decide branching bisimilarity, are the *alternating-bit protocol* (ABP), the *concurrent alternating-bit protocol* (CABP) and the *one-place buffer* (OPB). Each protocol allows sixteen different messages to be communicated.

The translation of [5] yields PBESs with an alternation depth of two. Every PBES is then instantiated to a BES by our tool. Finally, the BES is minimised using [13,14], and solved using the bigstep algorithm of Schewe.

[3] The tool PGSolver (version 2.0), available at http://www.tcs.ifi.lmu.de/pgsolver.

Table 3
Experimental results for deciding branching bisimilarity on the ABP, CABP and OPB models using both LTSs and PBESs.

(a) LTS sizes and generation times

Model	States	Transitions	Time (s)
ABP	755	976	<0.1
CABP	7184	28,960	0.6
OPB	17	32	<0.1

(b) BES sizes and times needed for PBES instantiation, BES solving and LTS comparison

		Time (s)		
Equivalence	BES size	PBES inst.	BES solve	LTS
ABP$\leftrightarrows_b$ OPB	26,987	1	<0.1	<0.1
OPB$\leftrightarrows_b$ CABP	358,322	10	0.3	<0.1
ABP$\leftrightarrows_b$ CABP	5,302,922	136	6.1	<0.1

We compare this approach to an LTS-based approach, in which the LTS for each model is generated by lps2lts, after which a branching-bisimilarity check on every pair of LTSs is performed by the mCRL2 tool ltscompare, that implements the algorithm from [37].

The results are listed in Table 3. The times for solving the BES consist only of the times needed for solving the corresponding parity game. The PBES-based approach turns out to be reasonably fast for the cases involving OPB. It is also demonstrated to be feasible in the case of ABP and CABP, but there it is significantly slower than the LTS-based approach. This is because the unsolved PBES (and therefore also its BES instantiation) is an explicitly represented over-approximation of the equivalence relation, whereas the LTS-based approach utilises more efficient ways of characterising such an equivalence relation, *viz.*, through the use of partitions of the state space. Our instantiation technique for this particular problem domain thus cannot be expected to rival the LTS-based approach; our experiments merely demonstrate the uniform nature of the PBES framework and instantiation as one of its strategies.

8. Conclusions

We have presented a verification methodology in which PBESs play a prominent role. They can be used for representing verification problems on infinite-state systems, most notably model checking for the first-order μ-calculus and equivalence checking for various bisimulation-like equivalences. Encoding these problems in PBESs can be done fully automatically. We presented the instantiation technique that aims to eliminate data from a PBES. Ultimately, this can yield a finite BES if all data sorts are finite, or by adopting an on-the-fly approach if some data sorts are countably infinite. A BES can be solved fully automatically using direct techniques or via a translation to parity games. Hence, our instantiation technique provides an important link in the chain of an automated, PBES-based verification process. Even though this process can never be fully automated, due to PBES solving being generally undecidable, our examples and experience with industrial case studies suggest that instantiation allows for full automation in most practical applications. Our examples involved complex model-checking and equivalence-checking problems, and demonstrated the efficacy of partial instantiation when solving model-checking problems on infinite-state systems manually. It should be noted that the equivalence-checking experiments we conducted cannot be expected to rival the state-of-the-art tooling for such problems, due to the way the problems must be encoded as PBESs. Nevertheless, the encodings, together with the instantiation strategy do show the versatility of our framework.

As a possible direction for future work, it may be interesting to combine instantiation with BES solving. By solving parts of the generated BES on-the-fly while instantiating a PBES, the construction of large BESs may be prevented effectively. The same goal can be pursued by minimising the PBES beforehand. For this, static analysis techniques along the lines of [38] and confluence reduction techniques must be developed. Finally, our prototype tooling can currently produce a rudimentary form of diagnostic information which is hard to interpret. This can be improved upon significantly by generating witnesses and counterexamples that are easier to understand. This has been achieved for BESs but only for the alternation-free segment [39]. We expect an extension to our general case to be non-trivial.

References

[1] J. Groote, T. Willemse, Parameterised boolean equation systems, Theor. Comput. Sci 343 (3) (2005) 332–369.
[2] R. Mateescu, Local model-checking of an alternation-free value-based modal mu-calculus, in: Proceedings of the Second International Workshop on VMCAI, 1998.
[3] J. Groote, R. Mateescu, Verification of temporal properties of processes in a setting with data, in: Proceedings of AMAST, Lecture Notes in Computer Science vol. 1548, Springer (1999) 74–90.
[4] J. Groote, T. Willemse, Model-checking processes with data, Sci. Comput. Program. 56 (3) (2005) 251–273.
[5] T. Chen, B. Ploeger, J.v.d. Pol, T. Willemse, Equivalence checking for infinite systems using parameterized boolean equation systems, in: Proceedings of the 18th International Conference on CONCUR, Lecture Notes in Computer Science vol. 4703, Springer (2007) 120–135.

[6] M. Gallardo, C. Joubert, P. Merino, Implementing influence analysis using parameterised boolean equation systems, in: Proceedings of the ISOLA'06, IEEE (2006).
[7] A.v. Dam, B. Ploeger, T. Willemse, Instantiation for parameterised boolean equation systems, in: Proceedings of the Fifth International Colloquium on Theoretical Aspects of Computing (ICTAC 2008), Lecture Notes in Computer Science vol. 5160, Springer (2008) 440–454.
[8] A. Mader, Verification of modal properties using boolean equation systems, Ph.D. Thesis, Technische Universität München, 1997.
[9] A. Mader, Verification of modal properties using infinite boolean equation systems, Tech. Rep. CSI-R9727, University of Nijmegen, Nijmegen, 1997.
[10] S. Schewe, Solving parity games in big steps, in: Proceedings of FSTTCS 2007, Lecture Notes in Computer Science vol. 4855, Springer (2007) 449–460.
[11] S. Schewe, An optimal strategy improvement algorithm for solving parity and payoff games, in: Proceedings of the 22nd International Workshop on Computer Science Logic (CSL 2008), Lecture Notes in Computer Science vol. 5213, Springer (2008) 369–384.
[12] J. Vöge, M. Jurdziński, A discrete strategy improvement algorithm for solving parity games, in: Proceedings of the CAV 2000, Lecture Notes in Computer Science vol. 1855, Springer (2000) 202–215.
[13] J. Keiren, T. Willemse, Bisimulation minimisations for boolean equation systems, in: HVC, Lecture Notes in Computer Science, vol. 6405, Springer, Heidelberg, 2011.
[14] M. Reniers, T. Willemse, Analysis of Boolean equation systems through structure graphs, in: B. Klin, P. Sobocinski (Eds.), SOS 2009, EPTCS, vol. 18, 2009, pp. 92–107.
[15] R. Mateescu, D. Thivolle, A model checking language for concurrent value-passing systems, in: Proceedings of FM 2008, Lecture Notes in Computer Science, vol. 5014, Springer-Verlag, 2008, pp. 148–164.
[16] R. Mateescu, CAESAR_SOLVE: a generic library for on-the-fly resolution of alternation-free boolean equation systems, STTT 8 (1) (2006) 37–56.
[17] J. van de Pol, M. Weber, A multi-core solver for parity games, ENTCS 220 (2) (2008) 19–34, pDMC 2008.
[18] M. Jurdziński, Small progress measures for solving parity games, in: STACS '00, Lecture Notes in Computer Science vol. 1770, Springer, Heidelberg (2000) 290–301.
[19] O. Friedmann, M. Lange, Solving parity games in practice, in: Proceedings of the ATVA, Lecture Notes in Computer Science vol. 5799, Springer (2009) 182–196.
[20] J. Groote, A. Mathijssen, M. Reniers, Y. Usenko, M.v. Weerdenburg, Analysis of distributed systems with mCRL2, in: M. Alexander, W. Gardner (Eds.), Process Algebra for Parallel and Distributed Processing, CRC Press, 2008, pp. 99–128.
[21] J. Baeten, W. Weijland, Process Algebra, Cambridge University Press, 1990.
[22] R. Milner, Communication and Concurrency, Prentice Hall, 1989.
[23] J. Groote, A. Ponse, The syntax and semantics of μCRL, in: Proceedings of the First Workshop on the Algebra of Communicating Processes (ACP 1994), Workshops in Computing, 1995, pp. 26–62.
[24] S. Luttik, Choice quantification in process algebra, Ph.D. Thesis, University of Amsterdam, April 2002.
[25] Y. Usenko, Linearization in μcrl, Ph.D. Thesis, Technische Universiteit Eindhoven, 2002.
[26] D. Kozen, Results on the propositional μ-calculus, Theor. Comput. Sci. 27 (1) (1983) 333–354.
[27] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, Pacific J. Math. 5 (2) (1955) 285–309.
[28] R. Milner, A Calculus of Communicating Systems, Springer, 1980.
[29] D. Park, Concurrency and automata on infinite sequences, in: Proceedings of the Fifth GI-Conference on Theoretical Computer Science, Lecture Notes in Computer Science, vol. 104, Springer, 1981, pp. 167–183.
[30] R.v. Glabbeek, W. Weijland, Branching time and abstraction in bisimulation semantics, J. ACM 43 (3) (1996) 555–600.
[31] H. Lin, Symbolic transition graph with assignment, in: CONCUR 1996, Lecture Notes in Computer Science vol. 1119, Springer, Heidelberg (1996) 50–65.
[32] S. Orzan, T. Willemse, Invariants for parameterised boolean equation systems, in: Proceedings of the 19th International Conference on Concurrency Theory (CONCUR 2008), Lecture Notes in Computer Science vol. 5201, Springer (2008) 187–202.
[33] H. Bekič, Programming languages and their definition, in: Lecture Notes in Computer Science, vol. 177, Springer-Verlag, 1984.
[34] M. van Weerdenburg, Efficient rewriting techniques, Ph.D. Thesis, Eindhoven University of Technology, 2009.
[35] J. Bradfield, Verifying Temporal Properties of Systems, Birkhäuser, 1992.
[36] J. Bradfield, C. Stirling, Modal logics and mu-calculi, in: J. Bergstra, A. Ponse, S. Smolka (Eds.), Handbook of Process Algebra, Elsevier, North-Holland, 2001, pp. 293–330 (Chapter 4).
[37] J. Groote, F. Vaandrager, An efficient algorithm for branching bisimulation and stuttering equivalence, in: Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP 1990), Lecture Notes in Computer Science, vol. 443, Springer, 1990, pp. 626–638.
[38] S. Orzan, J. Wesselink, T. Willemse, Static analysis techniques for parameterised boolean equation systems, in: S. Kowalewski, A. Philippou (Eds.), TACAS 2009, Lecture Notes in Computer Science, vol. 5505, Springer, 2009, pp. 230–245.
[39] R. Mateescu, Efficient diagnostic generation for boolean equation systems, in: S. Graf, M.I. Schwartzbach (Eds.), TACAS 2000, Lecture Notes in Computer Science, vol. 1785, Springer, 2000, pp. 251–265.